(Letter Head of the external organization)

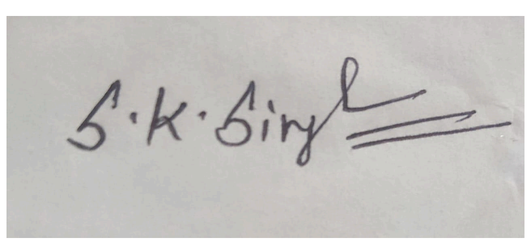
Annexure3b- Complete filing

TITLE: Hybrid Model for Real-Time Gesture Understanding

A. Full name	**Harisvardhan Singh Naghyal**
Mobile Number	6006968409
Email (personal)	harisvardhann@gmail.com
UID/Registration number	12319805
Address of Internal Inventors	Lovely Professional University, Punjab-144411, India
Signature (Mandatory)	

(Letter Head of the external organization)

Roll no.	44

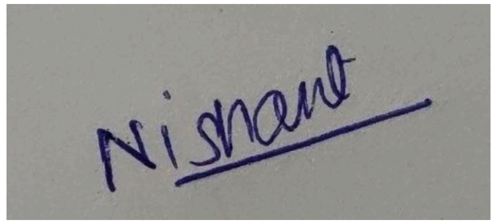
B. Full name	**Satyam Kumar Singh**
Mobile Number	8434475031
Email (personal)	satyamkumarsingh13990@gmail.com
UID/Registration number	12309199
Address of Internal Inventors	Lovely Professional University, Punjab-144411, India
Signature (Mandatory)	

(Letter Head of the external organization)

Roll no.	57

C. Full name	**Nishant Gangwar**
Mobile Number	12319998
Email (personal)	nishantgangwar499@gmail.com
UID/Registration number	12319998
Address of Internal Inventors	Lovely Professional University, Punjab-144411, India

(Letter Head of the external organization)

Signature (Mandatory)	
Roll no.	43

1. Project Overview

2. The AI-Powered Deadlock Detection Simulator is a software tool designed to model and analyze resource allocation scenarios in operating systems, specifically focusing on detecting and resolving deadlocks. Deadlocks are a critical issue in systems where multiple processes compete for limited resources, potentially leading to a standstill where no process can proceed. This project aims to provide a user-friendly graphical interface to simulate resource allocation graphs (RAGs), detect deadlocks using an AI-driven algorithm, and offer actionable resolution strategies. Built using Python, the simulator leverages the Tkinter library for its GUI and implements a modified banker's algorithm to identify deadlocks. The tool is intended for educational purposes, helping students and professionals understand deadlock concepts, and for system designers to test resource allocation strategies.

3. The simulator allows users to create processes and resources, define relationships (requests and allocations), visualize the system state, and analyze potential deadlocks. It also includes features like undo/redo functionality, state saving/loading, and a resolution guide to assist users in breaking deadlock cycles. The project integrates AI by using heuristic-based analysis to suggest optimal resolution steps, making it a valuable tool for both learning and practical applications.

2. Module-Wise Breakdown

The project is divided into several modules, each handling a specific aspect of the simulator. Below is a detailed breakdown:

2.1 Resource Allocation Graph (RAG) Module

This module forms the core of the system, managing the data structure that represents processes, resources, and their relationships. It includes:

- **Process Management**: Handles the creation and tracking of processes with unique identifiers (e.g., P1, P2).
- **Resource Management**: Manages resources, their total instances, and available instances.
- **Edge Management**: Tracks requests (process to resource) and allocations (resource to process) using a dictionary-based structure.
- **Deadlock Detection**: Implements a modified banker's algorithm to detect deadlocks by simulating resource allocation and checking for safe states.

2.2 Graphical User Interface (GUI) Module

The GUI module provides an interactive interface for users to visualize and manipulate the RAG. Key components include:

- **Canvas**: Displays processes as circles and resources as rectangles, with edges representing requests and allocations.
- **Control Panel**: Includes buttons and input fields to add nodes, create edges, and run simulations.
- **Status Bar**: Shows real-time updates on the system state, such as the number of processes, resources, and edges.

2.3 Deadlock Detection and Resolution Module

This module uses AI-driven techniques to detect deadlocks and suggest resolutions:

- **Detection**: Analyzes the RAG to identify cycles that indicate a deadlock, using a simulation-based approach.
- **Resolution Guide**: Provides step-by-step instructions to resolve deadlocks, such as releasing resources from specific processes.

- **AI Integration**: Employs heuristics to prioritize resolution steps, such as identifying the process holding the most critical resources.

2.4 State Management Module

This module ensures the system state can be saved, loaded, and manipulated:

- **Undo/Redo**: Tracks user actions (e.g., adding nodes, creating edges) and allows reversing or reapplying them.
- **State Export/Import**: Saves the RAG state as a JSON file and loads it back for future use.

2.5 Utility Module

Handles miscellaneous functionalities:

- **Node Positioning**: Automatically positions nodes on the canvas for better visualization.
- **Error Handling**: Displays user-friendly error messages for invalid actions (e.g., adding duplicate processes).

---

3. Functionalities

The simulator offers the following key functionalities:

1. **Process and Resource Creation**: Users can add processes and resources with customizable instances for resources.
2. **Edge Creation**: Supports request edges (process to resource) and allocation edges (resource to process), with adjustable counts.
3. **Visualization**: Displays the RAG on a canvas, with processes as blue circles, resources as green rectangles, and edges as arrows (red for requests, black for allocations).
4. **Deadlock Detection**: Runs an AI-driven algorithm to detect deadlocks and highlights affected processes and resources.
5. **Resolution Guidance**: Provides a detailed guide to resolve deadlocks, including suggestions for resource release.
6. **State Persistence**: Allows saving the current state to a JSON file and loading it later.
7. **Undo/Redo**: Enables users to revert or reapply actions, enhancing usability.

8. **Help System**: Offers a help dialog with instructions and shortcuts for ease of use.

---

4. Technology Used

Programming Languages:

- Python: Chosen for its simplicity, readability, and extensive library support. Python is used for both the backend logic and the GUI.

Libraries and Tools:

- Tkinter: A standard Python library for creating the graphical user interface. It provides widgets like buttons, canvases, and dialogs.
- ttk (Tkinter Themed Widgets): Enhances the GUI with modern, platform-native styling.
- json: Used for serializing and deserializing the RAG state to/from JSON format for saving and loading.
- collections.defaultdict: Simplifies the management of request and allocation edges by providing default values.
- sys, math: Utility libraries for system-level operations and mathematical calculations.

Other Tools:

- GitHub: Used for version control and collaboration, hosting the project repository.
- Visual Studio Code: The primary IDE for coding, debugging, and testing the application.

---

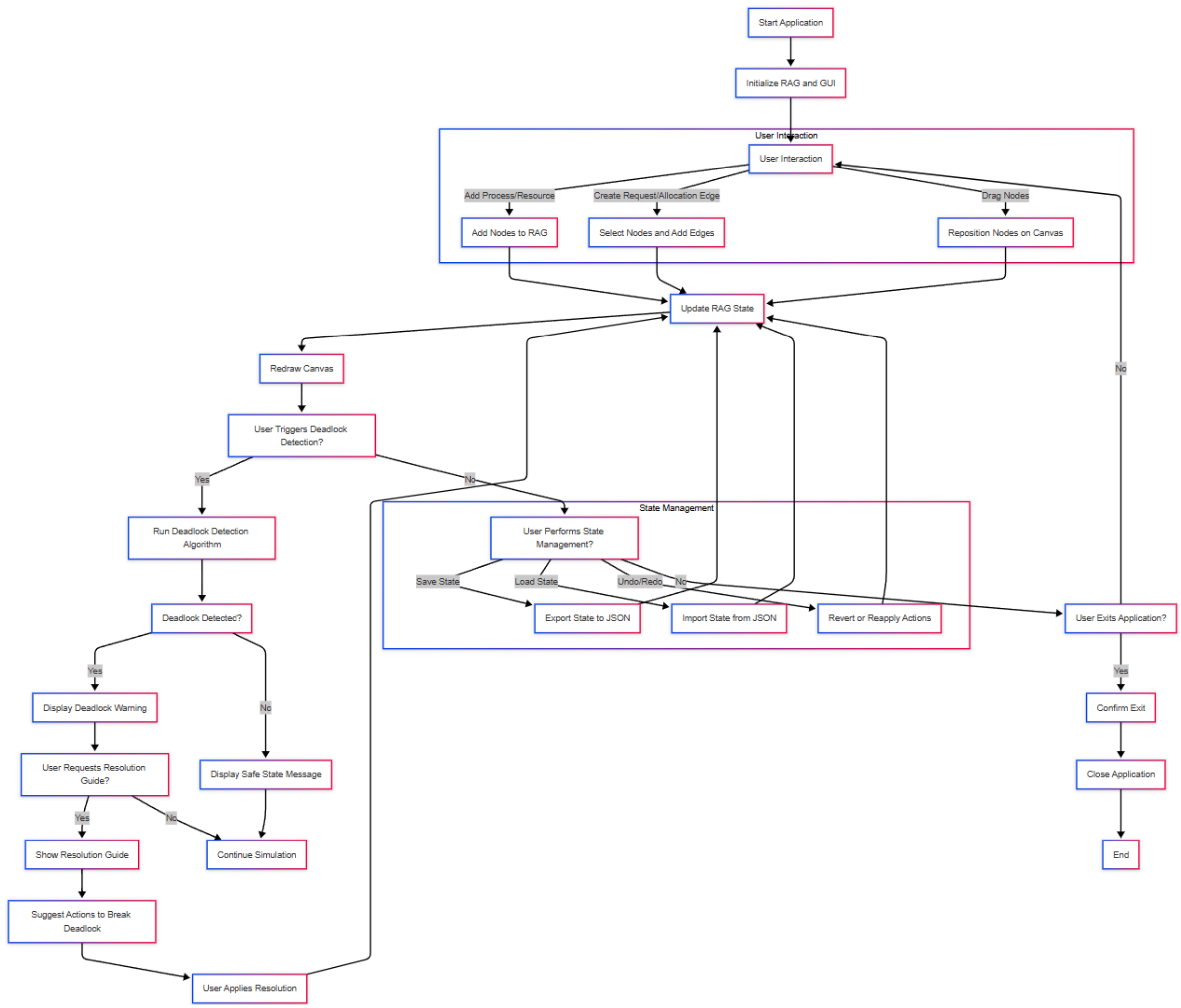
5. Flow Diagram

The flow of the simulator can be described as follows:

1. **Initialization**: The application starts, initializing the RAG and GUI.
2. **User Interaction**:
   - Add processes/resources via the control panel.
   - Create request/allocation edges by selecting nodes and specifying counts.
   - Drag nodes to reposition them on the canvas.

3. **State Update**: The RAG updates its internal state (processes, resources, edges) and redraws the canvas.
4. **Deadlock Detection**:
    - User triggers the detection process.
    - The system runs the banker's algorithm to check for deadlocks.
    - Results are displayed via a message box, and the status bar is updated.
5. **Resolution**:
    - If a deadlock is detected, the user can view the resolution guide.
    - The guide suggests actions like releasing resources to break the deadlock.
6. **State Management**:
    - Users can save/load the state or undo/redo actions as needed.
7. **Exit**: The application closes after user confirmation.

(Letter Head of the external organization)



---

6. Revision Tracking on GitHub

- **Repository Name:**

  AI-Deadlock-Simulator

- **GitHub Link:**

[https://github.com/harisvardhan/osproject]

(Note: The above is a placeholder. You would replace it with the actual repository link.)

The project is hosted on GitHub, where all changes are tracked. Key commits include:

- **Initial setup of the project structure and basic RAG implementation.**

- **Addition of the GUI using Tkinter.**
- **Implementation of the deadlock detection algorithm.**
- **Integration of undo/redo and state persistence features.**
- **Final testing and bug fixes.**

---

7. Conclusion and Future Scope

The AI-Powered Deadlock Detection Simulator successfully achieves its goal of providing an interactive tool for understanding and resolving deadlocks in resource allocation systems. The integration of AI-driven heuristics for deadlock resolution adds significant value, making the tool both educational and practical. The GUI is intuitive, and features like undo/redo and state persistence enhance user experience.

Future Scope:

1. Advanced AI Features: Incorporate machine learning to predict potential deadlocks before they occur based on historical data.
2. Multi-Threading Support: Simulate real-world scenarios with concurrent processes.
3. Cross-Platform Compatibility: Package the application for Windows, macOS, and Linux using tools like PyInstaller.
4. Web-Based Version: Develop a web version using frameworks like Flask or Django for broader accessibility.
5. Enhanced Visualization: Add animations to show the deadlock detection process step-by-step.

---

8. References

🎬 Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.

🎬 Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from https://docs.python.org/3/

🎬 Tkinter Documentation. (n.d.). Retrieved from https://docs.python.org/3/library/tkinter.html

- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. Pearson.

9.Appendix

A. AI-Generated Project Elaboration/Breakdown Report

The AI-Powered Deadlock Detection Simulator is a tool designed to simulate resource allocation scenarios in operating systems, focusing on deadlock detection and resolution. It uses a resource allocation graph (RAG) to model processes, resources, and their relationships. The project is divided into modules for RAG management, GUI, deadlock detection/resolution, state management, and utilities. The GUI, built with Tkinter, allows users to visualize the RAG, add nodes/edges, and detect deadlocks. The deadlock detection algorithm, inspired by the banker's algorithm, identifies unsafe states, while AI-driven heuristics suggest resolution steps. Features like undo/redo and state persistence enhance usability.

B. Problem Statement

In operating systems, deadlocks occur when multiple processes hold resources and wait for others, creating a cycle that prevents progress. Understanding and resolving deadlocks is crucial for system design and education. However, manually analyzing resource allocation scenarios is complex and error-prone. The goal of this project is to develop an AI-powered simulator that models resource allocation, detects deadlocks, and provides resolution guidance, making the process accessible and educational.

C. Solution/Code

Below is the complete code for the AI powered deadlock detection project

```
import tkinter as tk

from tkinter import ttk, messagebox, filedialog

from collections import defaultdict

import sys

import math
```

(Letter Head of the external organization)

```python
import json


class ResourceAllocationGraph:
    def __init__(self):
        self.processes = set()
        self.resources = {}
        self.allocations = defaultdict(int)
        self.requests = defaultdict(int)
        self.next_process_id = 1
        self.next_resource_id = 1

    def get_auto_process_name(self):
        while f"P{self.next_process_id}" in self.processes:
            self.next_process_id += 1
        return f"P{self.next_process_id}"

    def get_auto_resource_name(self):
        while f"R{self.next_resource_id}" in self.resources:
            self.next_resource_id += 1
        return f"R{self.next_resource_id}"

    def add_process(self, process_id=None):
        if not process_id:
            process_id = self.get_auto_process_name()
        if process_id in self.processes:
```

```
            raise ValueError(f"Process {process_id} already exists")
        self.processes.add(process_id)
        return process_id

    def add_resource(self, resource_id=None, instances=1):
        if not resource_id:
            resource_id = self.get_auto_resource_name()
        if resource_id in self.resources:
            raise ValueError(f"Resource {resource_id} already exists")
        self.resources[resource_id] = {'total': instances, 'available': instances}
        return resource_id

    def add_request(self, process, resource, count=1):
        if process not in self.processes or resource not in self.resources:
            raise ValueError("Invalid process or resource")
        self.requests[(process, resource)] += count

    def add_allocation(self, process, resource, count=1):
        if process not in self.processes or resource not in self.resources:
            raise ValueError("Invalid process or resource")
        if count > self.resources[resource]['available']:
            raise ValueError(f"Not enough instances available")
        self.allocations[(process, resource)] += count
        self.resources[resource]['available'] -= count
```

```python
def remove_allocation(self, process, resource, count=1):
    if (process, resource) in self.allocations:
        self.allocations[(process, resource)] -= count
        self.resources[resource]['available'] += count
        if self.allocations[(process, resource)] <= 0:
            del self.allocations[(process, resource)]


def detect_deadlock(self):
    work = {r: info['available'] for r, info in self.resources.items()}
    allocation = defaultdict(lambda: defaultdict(int))
    request = defaultdict(lambda: defaultdict(int))
    for (p, r), cnt in self.allocations.items():
        allocation[p][r] = cnt
    for (p, r), cnt in self.requests.items():
        request[p][r] = cnt
    finish = {p: False for p in self.processes}
    involved_resources = defaultdict(set)

    while True:
        found = False
        for p in self.processes:
            if not finish[p] and all(request[p][r] <= work[r] for r in self.resources):
                for r in self.resources:
                    work[r] += allocation[p][r]
                finish[p] = True
```

```
                found = True
            if not found:
                break

        deadlocked = [p for p, done in finish.items() if not done]
        if deadlocked:
            for p in deadlocked:
                for r in request[p]:
                    if request[p][r] > work[r]:
                        involved_resources[p].add(r)
        return len(deadlocked) > 0, deadlocked, involved_resources

    def get_deadlock_resolution_guide(self, deadlocked, involved_resources):
        if not deadlocked:
            return "No deadlock detected. The system is in a safe state."

        guide = "Deadlock Resolution Guide:\n\n"
        guide += f"Deadlocked Processes: {', '.join(deadlocked)}\n"
        guide += "Involved Resources and Requests:\n"
        for p in deadlocked:
            requests = [f"{r} ({self.requests[(p, r)]} requested)" for r in involved_resources[p]]
            allocations = [f"{r} ({self.allocations[(p, r)]} allocated)" for r in self.resources if (p, r) in self.allocations]
            guide += f"- {p}: Requests: {', '.join(requests) if requests else 'None'}, Allocations: {', '.join(allocations) if allocations else 'None'}\n"
```

```
        guide += "\nHow to Resolve:\n"
        guide += "1. Identify a process holding resources that others need.\n"
        guide += f"   - Suggestion: Release allocations from {deadlocked[0]}.\n"
        guide += "2. Release enough resources to break the cycle:\n"
        for r in involved_resources[deadlocked[0]]:
            if (deadlocked[0], r) in self.allocations:
                count = self.allocations[(deadlocked[0], r)]
                guide += f"   - Release {count} instance(s) of {r} from {deadlocked[0]} manually.\n"
        guide += "3. Adjust requests or add resources as needed.\n"
        return guide

    def export_state(self):
        return json.dumps({
            "processes": list(self.processes),
            "resources": self.resources,
            "allocations": dict(self.allocations),
            "requests": dict(self.requests)
        }, indent=4)

    def import_state(self, state_json):
        state = json.loads(state_json)
        self.processes = set(state["processes"])
        self.resources = state["resources"]
        self.allocations = defaultdict(int, state["allocations"])
```

```
        self.requests = defaultdict(int, state["requests"])


class RAGSimulator(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Resource Allocation Graph Simulator")
        self.geometry("1280x720")
        self.rag = ResourceAllocationGraph()
        self.selected_nodes = []
        self.edge_mode = None
        self.node_positions = {}
        self.dragging = None
        self.undo_stack = []
        self.redo_stack = []

        # UI Constants
        self.PROCESS_COLOR = "#4FC3F7"
        self.RESOURCE_COLOR = "#81C784"
        self.PROCESS_RADIUS = 30
        self.RESOURCE_SIZE = 70
        self.LEFT_MARGIN = 150
        self.RIGHT_MARGIN = 1100

        # Styling
        self.style = ttk.Style()
```

```
        self.style.theme_use('clam')
        self.style.configure("TButton", padding=5, font=('Helvetica', 10))
        self.style.configure("TLabel", font=('Helvetica', 11))
        self.style.configure("Header.TLabel", font=('Helvetica', 14, 'bold'))
        self.style.configure("Status.TLabel", font=('Helvetica', 9), background='#e0e0e0')

        self.setup_ui()
        self.setup_menu()
        self.protocol("WM_DELETE_WINDOW", self.on_close)

    def setup_menu(self):
        menubar = tk.Menu(self)
        file_menu = tk.Menu(menubar, tearoff=0)
        file_menu.add_command(label="New Graph", command=self.reset_graph,
accelerator="Ctrl+N")
        file_menu.add_command(label="Save State", command=self.export_state)
        file_menu.add_command(label="Load State", command=self.import_state)
        file_menu.add_separator()
        file_menu.add_command(label="Exit", command=self.on_close)
        menubar.add_cascade(label="File", menu=file_menu)

        edit_menu = tk.Menu(menubar, tearoff=0)
        edit_menu.add_command(label="Undo", command=self.undo, accelerator="Ctrl+Z")
        edit_menu.add_command(label="Redo", command=self.redo, accelerator="Ctrl+Y")
        menubar.add_cascade(label="Edit", menu=edit_menu)
```

```
        menubar.add_command(label="Help", command=self.show_help)
        self.config(menu=menubar)

    def setup_ui(self):
        main_frame = ttk.Frame(self, padding=10)
        main_frame.pack(fill=tk.BOTH, expand=True)

        control_pane = ttk.PanedWindow(main_frame, orient=tk.VERTICAL)
        control_pane.pack(side=tk.LEFT, fill=tk.Y, padx=(0, 10))

        node_frame = ttk.LabelFrame(control_pane, text="Add Nodes", padding=8)
        control_pane.add(node_frame, weight=1)

        ttk.Button(node_frame, text="New Process",
command=self.add_process).pack(fill=tk.X, pady=2)
        ttk.Button(node_frame, text="New Resource",
command=self.add_resource).pack(fill=tk.X, pady=2)
        self.instances_var = tk.IntVar(value=1)
        ttk.Label(node_frame, text="Resource Instances:").pack(pady=(5, 2))
        ttk.Spinbox(node_frame, from_=1, to=10, textvariable=self.instances_var,
                    width=5).pack()

        edge_frame = ttk.LabelFrame(control_pane, text="Edges", padding=8)
        control_pane.add(edge_frame, weight=1)
```

```
self.count_var = tk.IntVar(value=1)
ttk.Label(edge_frame, text="Edge Count:").pack(pady=(0, 2))
ttk.Spinbox(edge_frame, from_=1, to=5, textvariable=self.count_var,
          width=5).pack(pady=2)
ttk.Button(edge_frame, text="Request Edge",
         command=lambda: self.set_edge_mode("request")).pack(fill=tk.X, pady=2)
ttk.Button(edge_frame, text="Allocation Edge",
         command=lambda: self.set_edge_mode("allocation")).pack(fill=tk.X, pady=2)
ttk.Button(edge_frame, text="Clear Selection",
         command=self.clear_selection).pack(fill=tk.X, pady=2)

sim_frame = ttk.LabelFrame(control_pane, text="Simulation", padding=8)
control_pane.add(sim_frame, weight=1)
ttk.Button(sim_frame, text="Check Deadlock",
         command=self.detect_deadlock).pack(fill=tk.X, pady=2)
ttk.Button(sim_frame, text="Resolution Guide",
         command=self.show_resolution_guide).pack(fill=tk.X, pady=2)

self.canvas = tk.Canvas(main_frame, bg='white', highlightthickness=0)
self.canvas.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
self.canvas.bind("<Button-1>", self.on_click)
self.canvas.bind("<B1-Motion>", self.on_drag)
self.canvas.bind("<ButtonRelease-1>", self.on_release)

self.status = ttk.Label(main_frame, text="Ready", style="Status.TLabel",
```

```
                    relief=tk.SUNKEN, anchor='w', padding=4)
        self.status.pack(side=tk.BOTTOM, fill=tk.X)

        self.bind_all("<Control-z>", lambda e: self.undo())
        self.bind_all("<Control-y>", lambda e: self.redo())
        self.bind_all("<Control-n>", lambda e: self.reset_graph())

    def show_help(self):
        messagebox.showinfo("Help",
            "Resource Allocation Graph Simulator\n\n"
            "Shortcuts:\n"
            "Ctrl+Z: Undo\n"
            "Ctrl+Y: Redo\n"
            "Ctrl+N: New Graph\n\n"
            "Usage:\n"
            "1. Add processes and resources\n"
            "2. Select two nodes to create edges\n"
            "3. Request: Process → Resource (red)\n"
            "4. Allocation: Resource → Process (black)\n"
            "5. Drag nodes to reposition\n"
            "6. Check deadlocks and use 'Resolution Guide' for manual resolution tips")

    def export_state(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".json",
                                                  filetypes=[("JSON files", "*.json")])
```

```
    if file_path:
        with open(file_path, 'w') as f:
            f.write(self.rag.export_state())
        self.status.config(text="State saved")

def import_state(self):
    file_path = filedialog.askopenfilename(filetypes=[("JSON files", "*.json")])
    if file_path:
        with open(file_path, 'r') as f:
            self.push_undo_action('import', self.rag.export_state())
            self.rag.import_state(f.read())
        self.reposition_nodes()
        self.update_display()
        self.status.config(text="State loaded")

def set_edge_mode(self, mode):
    self.edge_mode = mode
    self.selected_nodes = []
    self.status.config(text=f"Select nodes for {mode} edge")

def update_display(self):
    self.canvas.delete("all")
    self.draw_edges()
    self.draw_nodes()
    self.draw_selection()
```

```
    self.status.config(text=f"P: {len(self.rag.processes)} | R: {len(self.rag.resources)} | "
                       f"Edges: {len(self.rag.allocations) + len(self.rag.requests)}")

def draw_edges(self):
    for (p, r), count in self.rag.requests.items():
        if count > 0 and p in self.node_positions and r in self.node_positions:
            self.draw_edge(p, r, "request", count)
    for (p, r), count in self.rag.allocations.items():
        if count > 0 and p in self.node_positions and r in self.node_positions:
            self.draw_edge(r, p, "allocation", count)

def draw_edge(self, from_node, to_node, edge_type, count):
    x1, y1 = self.node_positions[from_node]
    x2, y2 = self.node_positions[to_node]
    color = '#EF5350' if edge_type == "request" else '#424242'
    arrow = tk.FIRST if edge_type == "request" else tk.LAST
    mid_x, mid_y = (x1 + x2) / 2, (y1 + y2) / 2
    self.canvas.create_line(x1, y1, x2, y2, fill=color, width=2, arrow=arrow)
    self.canvas.create_text(mid_x, mid_y - 10, text=str(count), fill=color,
                       font=('Helvetica', 10, 'bold'))

def draw_nodes(self):
    for p in self.rag.processes:
        if p in self.node_positions:
            x, y = self.node_positions[p]
```

```
            self.canvas.create_oval(x-self.PROCESS_RADIUS, y-self.PROCESS_RADIUS,
                                    x+self.PROCESS_RADIUS, y+self.PROCESS_RADIUS,
                                    fill=self.PROCESS_COLOR, outline='#0277BD', width=2,
                                    tags=('node', p))
            self.canvas.create_text(x, y, text=p, font=('Helvetica', 12, 'bold'), tags=('text', p))
        for r in self.rag.resources:
            if r in self.node_positions:
                x, y = self.node_positions[r]
                avail = self.rag.resources[r]['available']
                fill = self.RESOURCE_COLOR if avail > 0 else '#EF9A9A'
                self.canvas.create_rectangle(x-self.RESOURCE_SIZE/2,
y-self.RESOURCE_SIZE/2,
                                        x+self.RESOURCE_SIZE/2, y+self.RESOURCE_SIZE/2,
                                        fill=fill, outline='#2E7D32', width=2, tags=('node', r))
                self.canvas.create_text(x, y-10, text=r, font=('Helvetica', 12, 'bold'))
                self.canvas.create_text(x, y+10, text=f"{avail}/{self.rag.resources[r]['total']}",
                                    font=('Helvetica', 10))

    def draw_selection(self):
        for node in self.selected_nodes:
            x, y = self.node_positions[node]
            size = self.PROCESS_RADIUS if node in self.rag.processes else
self.RESOURCE_SIZE/2
            self.canvas.create_oval(x-size-5, y-size-5, x+size+5, y+size+5,
                                outline='#FFB300', width=2, dash=(4, 2))
```

(Letter Head of the external organization)

```python
def on_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    tags = self.canvas.gettags(item)
    if 'node' in tags or 'text' in tags:
        node = tags[1]
        if self.edge_mode:
            if node not in self.selected_nodes:
                self.selected_nodes.append(node)
                if len(self.selected_nodes) == 2:
                    self.create_edge()
        else:
            self.dragging = node
            self.drag_offset = (event.x - self.node_positions[node][0],
                                event.y - self.node_positions[node][1])
        self.update_display()

def on_drag(self, event):
    if self.dragging:
        self.node_positions[self.dragging] = (event.x - self.drag_offset[0],
                                              event.y - self.drag_offset[1])
        self.update_display()

def on_release(self, event):
    self.dragging = None
```

```
def create_edge(self):
    try:
        if len(self.selected_nodes) != 2:
            return
        n1, n2 = self.selected_nodes
        count = self.count_var.get()
        if self.edge_mode == "request" and n1 in self.rag.processes and n2 in
self.rag.resources:
            self.rag.add_request(n1, n2, count)
            self.push_undo_action('request', n1, n2, count)
        elif self.edge_mode == "allocation" and n1 in self.rag.resources and n2 in
self.rag.processes:
            self.rag.add_allocation(n2, n1, count)
            self.push_undo_action('allocation', n2, n1, count)
        else:
            raise ValueError("Invalid edge direction")
        self.clear_selection()
        self.update_display()
    except ValueError as e:
        messagebox.showerror("Error", str(e))
        self.clear_selection()


def push_undo_action(self, action_type, node1=None, node2=None, count=0,
state=None):
    self.undo_stack.append({
        'type': action_type, 'node1': node1, 'node2': node2,
```

```
        'count': count, 'state': state, 'positions': self.node_positions.copy()
    })
    self.redo_stack.clear()

def undo(self):
    if not self.undo_stack:
        return
    action = self.undo_stack.pop()
    if action['type'] == 'request':
        self.rag.requests[(action['node1'], action['node2'])] -= action['count']
        if self.rag.requests[(action['node1'], action['node2'])] <= 0:
            del self.rag.requests[(action['node1'], action['node2'])]
    elif action['type'] == 'allocation':
        self.rag.remove_allocation(action['node2'], action['node1'], action['count'])
    elif action['type'] == 'add_process':
        self.rag.processes.remove(action['node1'])
        del self.node_positions[action['node1']]
    elif action['type'] == 'add_resource':
        del self.rag.resources[action['node1']]
        del self.node_positions[action['node1']]
    elif action['type'] == 'import':
        self.rag.import_state(action['state'])
        self.node_positions = action['positions']
    self.redo_stack.append(action)
    self.update_display()
```

(Letter Head of the external organization)

```
def redo(self):
    if not self.redo_stack:
        return
    action = self.redo_stack.pop()
    if action['type'] == 'request':
        self.rag.add_request(action['node1'], action['node2'], action['count'])
    elif action['type'] == 'allocation':
        self.rag.add_allocation(action['node2'], action['node1'], action['count'])
    elif action['type'] == 'add_process':
        self.rag.add_process(action['node1'])
        self.node_positions[action['node1']] = action['positions'][action['node1']]
    elif action['type'] == 'add_resource':
        self.rag.add_resource(action['node1'], action['count'])
        self.node_positions[action['node1']] = action['positions'][action['node1']]
    elif action['type'] == 'import':
        curr_state = self.rag.export_state()
        self.rag.import_state(action['state'])
        action['state'] = curr_state
    self.undo_stack.append(action)
    self.update_display()

def add_process(self):
    try:
        p_id = self.rag.add_process()
```

```
        self.node_positions[p_id] = (self.LEFT_MARGIN, len(self.rag.processes) * 100)
        self.push_undo_action('add_process', p_id)
        self.update_display()
    except ValueError as e:
        messagebox.showerror("Error", str(e))

def add_resource(self):
    try:
        r_id = self.rag.add_resource(instances=self.instances_var.get())
        self.node_positions[r_id] = (self.RIGHT_MARGIN, len(self.rag.resources) * 100)
        self.push_undo_action('add_resource', r_id, count=self.instances_var.get())
        self.update_display()
    except ValueError as e:
        messagebox.showerror("Error", str(e))

def clear_selection(self):
    self.selected_nodes = []
    self.edge_mode = None
    self.update_display()

def detect_deadlock(self):
    has_deadlock, processes, involved_resources = self.rag.detect_deadlock()
    if has_deadlock:
        messagebox.showwarning("Deadlock", f"Deadlocked: {', '.join(processes)}\n"
                               f"Involved Resources: {dict(involved_resources)}")
```

```
            self.status.config(text="Deadlock detected!")
        else:
            messagebox.showinfo("Safe", "No deadlock detected")
            self.status.config(text="System safe")
        return has_deadlock, processes, involved_resources

    def show_resolution_guide(self):
        has_deadlock, processes, involved_resources = self.rag.detect_deadlock()
        guide = self.rag.get_deadlock_resolution_guide(processes, involved_resources)
        if has_deadlock:
            messagebox.showinfo("Deadlock Resolution Guide", guide)
        else:
            messagebox.showinfo("No Deadlock", guide)

    def reset_graph(self):
        if messagebox.askyesno("Reset", "Clear all data?"):
            self.push_undo_action('import', self.rag.export_state())
            self.rag = ResourceAllocationGraph()
            self.node_positions = {}
            self.clear_selection()
            self.redo_stack.clear()
            self.update_display()
            self.status.config(text="Graph reset")

    def reposition_nodes(self):
```

```
        self.node_positions.clear()
        for i, p in enumerate(self.rag.processes):
            self.node_positions[p] = (self.LEFT_MARGIN, (i + 1) * 100)
        for i, r in enumerate(self.rag.resources):
            self.node_positions[r] = (self.RIGHT_MARGIN, (i + 1) * 100)

    def on_close(self):
        if messagebox.askokcancel("Quit", "Exit application?"):
            self.destroy()

if __name__ == "__main__":
    app = RAGSimulator()
    app.mainloop()
```