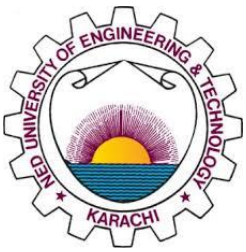



# Mafia Game Project

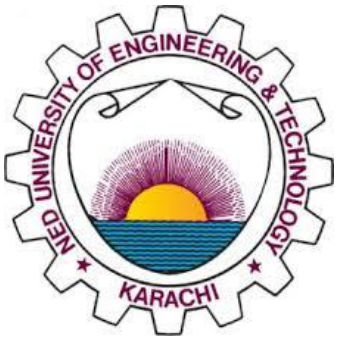
Python GUI Application

**By :- Hadiya Ayeshah (SE-25-016)**

# 📘 MAFIA GAME – PROJECT REPORT

**(Complex Engineering Problem -CEP)**

**(Complex Engineering Activity -CEA)**




| Element | Content |
|---|---|
| Title | MAFIA GAME PROJECT |
| Subtitle | A Python GUI Application |
| Submitted By: | Hadiya Ayeshah (SE-25016) |
| Course Name | Programming Fundamentals |
| Course Code | SE-101 |
| Submitted To: | Miss Asma |
| Semester | Fall-2025 |
| Date | 18 November, 2025 |

# 1. Problem Statement

Turn-based multiplayer games like Mafia involve multiple players performing different roles with specific actions, creating complex interactions and conditional logic that can be difficult to manage manually. Players often face confusion regarding game state transitions, role assignments, and the outcomes of their actions, which can reduce engagement and fairness. Existing implementations may lack a clear interface, real-time feedback, or robust error handling, resulting in a suboptimal gameplay experience.

This project addresses these challenges by developing a Python-based graphical Mafia game for 4 players, featuring an intuitive GUI that clearly communicates game states, dynamically assigns roles, and manages Night and Day cycles. The system accurately processes player actions such as assassinations, investigations, and voting while ensuring smooth, uninterrupted gameplay. Additionally, the design emphasizes modularity and scalability, enabling future expansions, additional roles, and customizable game rules. The project thus provides both a practical solution to gameplay challenges and an opportunity to demonstrate skills in GUI design, event-driven programming, and complex game logic management.

## The main requirements were:

- Create a **GUI-based interactive game** using Python and Tkinter.
- Allow **four players** to enter their names.
- Randomly assign **one Mafia, one Doctor, and two Civilians**.
- Execute decisions made at night (attack and save).
- Casts vote to lynch out the person you suspect is the Mafia during the day.
- Display a **game summary**.
- Include **graphic improvements**, animations, and a clean layout.

## 2. Table of Contents:-

## 3. Project Objective

The objective of this project is to design and implement a fully functional, turn-based Mafia game using Python and a Graphical User Interface (GUI). The project aims to create an interactive system that automates all essential components of Mafia gameplay, including dynamic role assignment, state transitions between Night and Day phases, and the processing of player actions such as assassinations, investigations, and voting.

A key objective is to develop a clear, intuitive, and visually engaging interface that enhances user interaction and minimizes confusion during gameplay. The system is designed to provide real-time feedback, visual indicators, and well-structured prompts to ensure players remain fully aware of their roles, available actions, and the current game state.

The project also prioritizes modular and scalable software design, enabling future enhancements such as expanded player capacity, additional character roles, configurable rules, and improved animations or graphics. Through this implementation, the project aims to demonstrate proficiency in event-driven programming, GUI development, logical problem-solving, and game state management. Ultimately, the objective is to deliver a polished, user-friendly game that accurately simulates the strategic depth and suspense of Mafia while maintaining code clarity, robustness, and maintainability.

## 4. *Methodology*

The methodology for developing the Mafia Game application followed a structured, iterative approach that ensured clarity in design, accuracy in game logic, and a user-friendly graphical interface. The development process was divided into several key phases:

### 1. Requirements Analysis

The project began by identifying the essential gameplay mechanics of Mafia, including:

- The number of players and their roles
- Night and Day cycles
- Action sequences (kill, investigate, vote)
- Win conditions and state transitions

## 2. System Design

A modular software design approach was adopted to simplify complexity and streamline maintainability.

### a. Architecture Design

The system was structured into the following primary components:

- **Role Management Module**: Handles dynamic role assignment.
- **Game Logic Module**: Controls Night/Day transitions, action resolution, and win condition evaluation.
- **GUI Module**: Manages all visual elements, user interactions, and event-driven commands.

### b. Flow Design

A logical flowchart was drafted to visualize game progression, including:

- Game initialization
- Night action processing
- Daytime voting
- Victory condition checks

## 3. GUI Development

The Tkinter library was used to build an interactive and responsive user interface.

**Key steps included:**

- Designing the layout (buttons, labels, dialogs, and frames)
- Implementing color schemes for clarity (e.g., Night vs. Day themes)
- Adding dynamic message displays for player feedback
- Ensuring accessibility and intuitive navigation

**4. Game Logic Implementation**

The core mechanics were implemented using event-driven programming.

Tasks included:

- Encoding role behaviors (Mafia, Detective, Civilian, etc.)
- Programming conditional sequences for Night and Day actions
- Validating player inputs and preventing illegal actions
- Handling edge cases such as ties, early elimination, or skipped votes

**5. Testing and Debugging**

Multiple levels of testing were performed:

- **Unit Testing:** To validate individual modules and functions.
- **Integration Testing:** To ensure smooth coordination between GUI and logic components.
- **User Testing:** By simulating multiple game rounds to observe gameplay flow, identify bugs, and measure user experience.

**6. Refinement and Optimization**

Based on test results, several refinements were made:

- Improving UI clarity and responsiveness
- Enhancing role notifications and guidance prompts
- Streamlining game logic for performance
- Ensuring the system is future-proof with modular design for expansion

**7. Documentation**

The final phase involved preparing:

- User documentation
- Source code comments
- Project report sections including objective, problem statement, methodology, and

## 5. Distinguishing Features of the Project:-

Mafia Game stands out due to the following distinctive features:

**★ Fully Graphical Interface**

The game uses Tkinter to create a clean, modern GUI with:

- Custom colors
- Enhanced layout spacing
- Title screen
- Rounded-style buttons

### ★ Interactive Player Setup

All 4 players:

- Enter names
- Are assigned roles from drop-down menu by the game master
- View instructions clearly

### ★ Role-Based Logic

The game handles:

- Mafia choosing a target
- Doctor choosing someone to save
- Automatic evaluation of whether someone dies

### ★ Automatic Game Summary

The final screen displays:

- Who was attacked
- Who was saved
- Who died (if any)
- Which side won

**★ Improved Game Flow**

If:

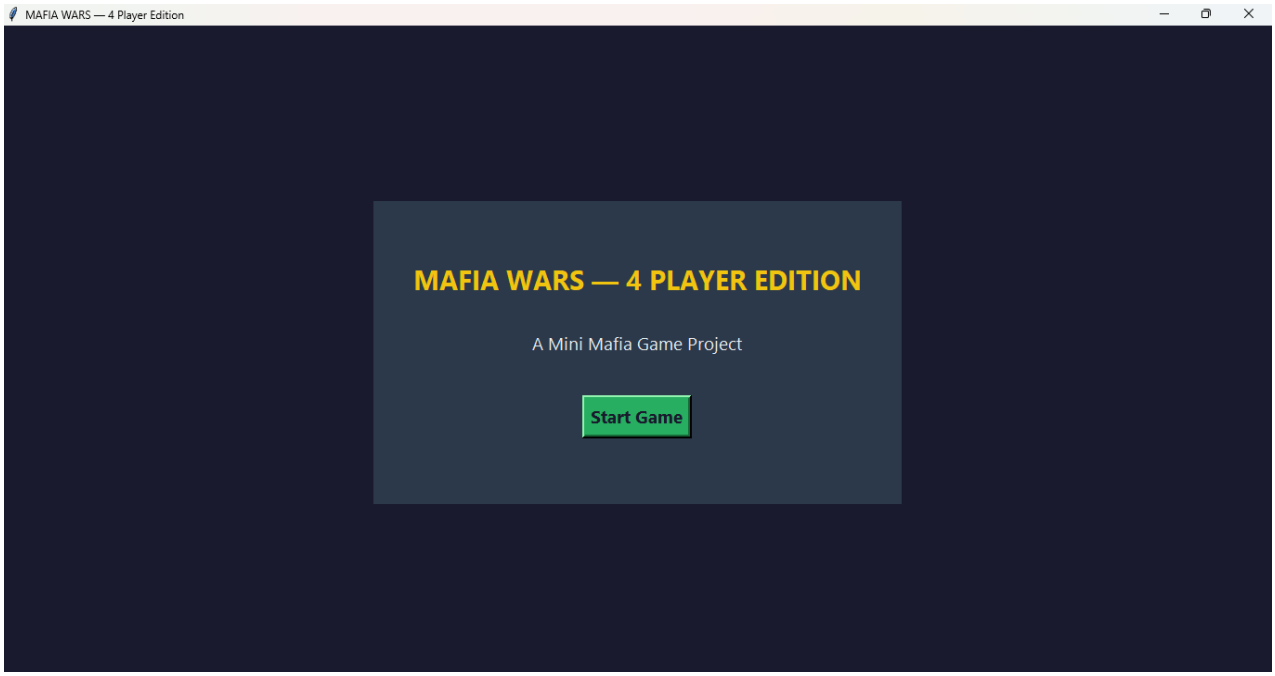
Doctor saves the correct target → No one dies → Game ends immediately.

**★ Replayable & Visually Enhanced**

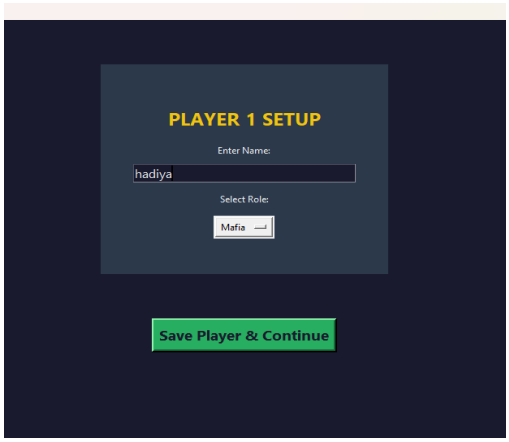
- Title screen
- Custom styling
- Interaction through GUI

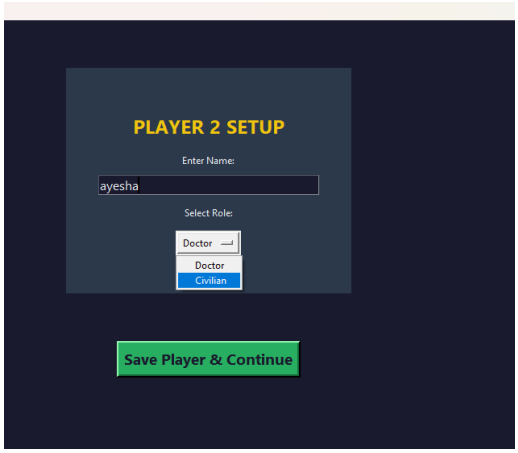
# 6. Screenshots of the Main Features

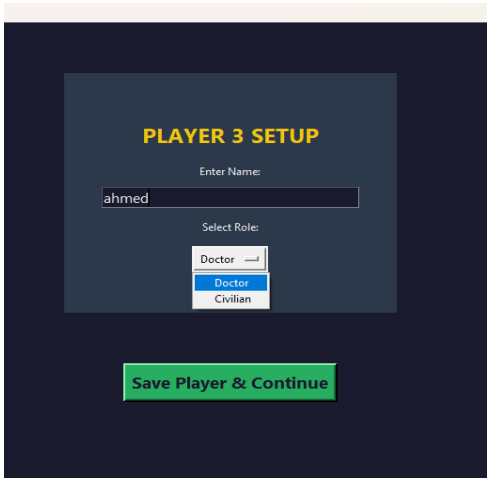
1. **Title Screen** – The starting page

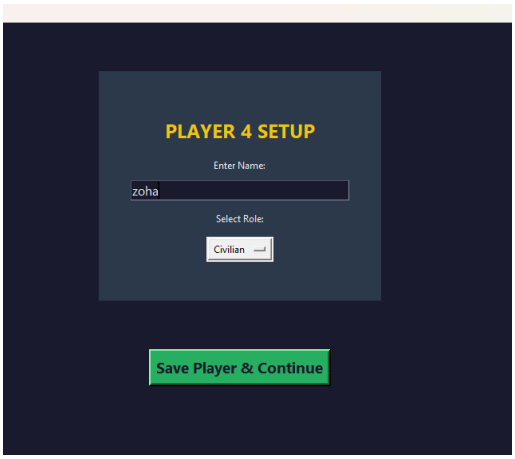


2. **Role Assignment Screens:-**

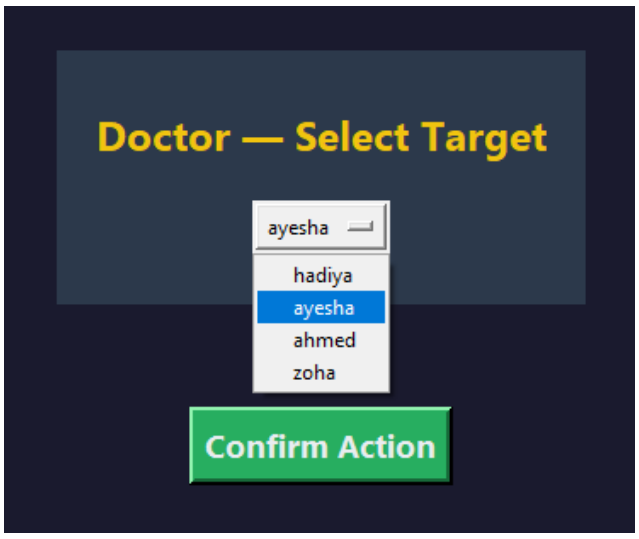


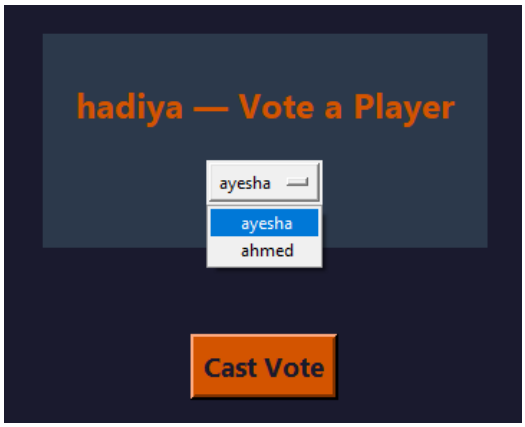



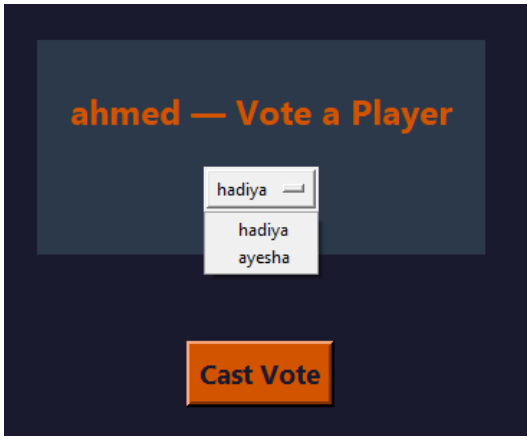

**3. Night Selection (Mafia + Doctor)**

## 4. Day Actions (Voting)

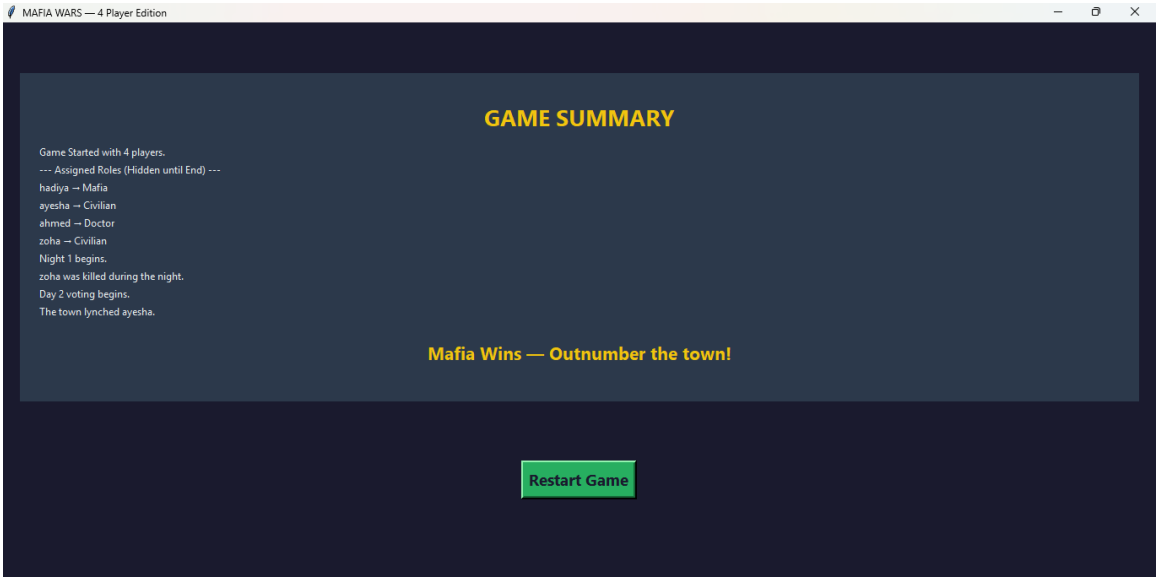







Everybody votes except the person who was killed by the Mafia and the Doctor failed to save.

## 5. Final Game Summary Screen



# 7. Source Code of the Main Features:-

## I. Player Setup Logic

```
    # Player Setup

    def setup_ui(self):  2 usages
        self.clear_ui()
        self.setup_step = 0
        self.setup_player_data = []
        self.setup_role_counts = {"Mafia": 0, "Doctor": 0, "Civilian": 0}
        self.next_setup_step()

    def next_setup_step(self):  2 usages
        self.clear_ui()

        if self.setup_step >= self.TOTAL_PLAYERS:
            self.finalize_setup()
            return

        frame = tk.Frame(self.main_frame, bg=BG_FRAME, padx=40, pady=40, width=600, height=400)
        frame.pack(pady=40)

        tk.Label(
            frame,
            text=f"PLAYER {self.setup_step + 1} SETUP",
            font=("Segoe UI", 18, "bold"),
            bg=BG_FRAME,
            fg=ACCENT_HEADER
        ).pack(pady=14)
```

## II. Role Assignment

```
def __init__(self):
    super().__init__()
    self.title("MAFIA WARS – 4 Player Edition")
    self.geometry("650x750")
    self.configure(bg=BG_MAIN)

    self.players = {}
    self.status = {}
    self.living_players = []
    self.phase = "Setup"
    self.round_number = 1
    self.summary_log = []

    # Setup trackers
    self.setup_step = 0
    self.setup_player_data = []
    self.setup_role_counts = {"Mafia": 0, "Doctor": 0, "Civilian": 0}
```

```
def process_setup_step(self):  1 usage
    name = self.name_entry.get().strip()
    role = self.role_var.get()

    if not name:
        messagebox.showerror( title: "Error",  message: "Name cannot be em
        return

    if name in [p[0] for p in self.setup_player_data]:
        messagebox.showerror( title: "Error",  message: "Name must be uniq
        return

    self.setup_player_data.append((name, role))
    self.setup_role_counts[role] += 1
    self.setup_step += 1
    self.next_setup_step()

def finalize_setup(self):  1 usage
    self.players = {name: role for name, role in self.setup_player_data
    self.status = {name: 'alive' for name in self.players}
    self.living_players = list(self.players.keys())

    self.summary_log.append("Game Started with 4 players.")
    self.summary_log.append("--- Assigned Roles (Hidden until End) ---
    for n, r in self.players.items():
```

## III. Night Cycle Logic:

```
# ------------------------------------------------
def start_night_phase(self):  2 usages
    self.summary_log.append(f"Night {self.round_number} begins.")
    self.night_kill_target = None
    self.night_save_target = None
    self.pending_roles = ["Mafia", "Doctor"]
    self.night_action_ui()

def night_action_ui(self):  2 usages
    self.clear_ui()

    if not self.pending_roles:
        self.process_night_results()
        return

    role = self.pending_roles.pop(0)

    frame = tk.Frame(self.main_frame, bg=BG_FRAME, padx=20, pady=20)
    frame.pack(pady=40)
```

```
def record_night_action(self, role):  1 usage
    target = self.target_var.get()

    if role == "Mafia":
        self.night_kill_target = target

    if role == "Doctor":
        self.night_save_target = target

    self.night_action_ui()

def process_night_results(self):  1 usage
    kill = self.night_kill_target
    save = self.night_save_target

    # doctor saves the target → end game immediately
    if kill == save:
        self.summary_log.append(f"Doctor saved {kill}. No one died.")
        self.show_end_screen("No one died — The Doctor saved the target. Game Over.")
        return

    # someone dies
    self.status[kill] = 'dead'
    self.living_players.remove(kill)
    self.summary_log.append(f"{kill} was killed during the night.")
```

## IV. Day Phase:

```
# ------------------------------------------------
def start_day_phase(self):  1 usage
    self.summary_log.append(f"Day {self.round_number} voting begins.")
    self.day_votes = {}
    self.voter_index = 0
    self.voters = list(self.living_players)
    self.day_vote_ui()

def day_vote_ui(self):  2 usages
    self.clear_ui()

    if self.voter_index >= len(self.voters):
        self.process_day_results()
        return

    voter = self.voters[self.voter_index]

    frame = tk.Frame(self.main_frame, bg=BG_FRAME, padx=20, pady=20)
    frame.pack(pady=40)

    tk.Label(
        frame,
        text=f"{voter} — Vote a Player",
        font=("Segoe UI", 18, "bold"),
        bg=BG_FRAME,
        fg=ACCENT_VOTE
    ).pack(pady=10)
```

```
    targets = [p for p in self.living_players if p != voter]

    self.vote_var = tk.StringVar(self)
    self.vote_var.set(targets[0])

    tk.OptionMenu(frame, self.vote_var, *values: *targets).pack(pady=10)

    self.create_button(
        self.main_frame,
        text="Cast Vote",
        cmd=self.record_day_vote,
        bg=ACCENT_VOTE,
        fg=BG_MAIN
    ).pack(pady=20)

def record_day_vote(self):  1 usage
    target = self.vote_var.get()
    self.day_votes[target] = self.day_votes.get(target, 0) + 1
    self.voter_index += 1
    self.day_vote_ui()

def process_day_results(self):  1 usage
    lynch = max(self.day_votes, key=self.day_votes.get)
```

## V. Game Summary Display

```
tk.Label(
    frame,
    text="GAME SUMMARY",
    font=("Segoe UI", 20, "bold"),
    bg=BG_FRAME,
    fg=ACCENT_HEADER
).pack(pady=10)

for line in self.summary_log:
    tk.Label(
        frame,
        text=line,
        bg=BG_FRAME,
        fg=TEXT_LIGHT,
        anchor="w"
    ).pack(fill="x")

tk.Label(
    frame,
    text=msg,
    font=("Segoe UI", 16, "bold"),
    bg=BG_FRAME,
    fg=ACCENT_HEADER
).pack(pady=20)
```

## 6. Future Expansions

We plan to improve the game with:

⭐ **Advanced Graphics**

Use images, icons, and rounded buttons with custom themes.

⭐ **More Roles**

Introduce Detective, Serial Killer, Jester, etc.

⭐ **Full Animation**

Fade effects, smooth transitions between screens.

⭐ **Scoreboard System**

Track wins and losses for players across matches.

## 7. Conclusion:

The Mafia Game project successfully demonstrates the design and implementation of a fully interactive, turn-based multiplayer game using Python and a Graphical User Interface (GUI). Through this project, a complete game cycle including role assignment, Night and Day phases, player actions, voting, and win-condition evaluation was automated in an engaging and visually structured environment.

The system effectively manages complex game logic by incorporating event-driven programming, clear state transitions, and modular code organization. Features such as dropdown role selection, targeted actions, dynamic UI updates, and a comprehensive end-game summary contribute to an intuitive user experience. The GUI design enhances clarity and player involvement, ensuring that every stage of the game is understandable and easy to follow.

The project also strengthens core programming skills, including GUI development with Tkinter, data structures, object-oriented programming, and logical decision-making. This foundation allows for future features such as additional roles, animations, sound effects, more players, and custom rule configurations.

Overall, the project achieves its primary objective of delivering a functional and enjoyable digital version of the Mafia game while providing substantial practical experience in developing interactive, state-driven desktop applications.