UCS2504- Foundations Of Artificial Intelligence

CLUEDO

(Assignment – 2/Report)

Submitted By

Prithivirajan D - 3122225001099

Rithekha K – 3122225001106

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Kalavakkam – 603 110

October 2024

TABLE OF CONTENTS:

Cluedo Game: AI vs Player Implementation

1) PROBLEM STATEMENT:

Clue Game is a strategic deduction game where the player competes against an AI to deduce the correct suspect, weapon, and location involved in a fictional crime. Similar to the classic game Clue, each player attempts to identify the correct combination of "murderer," "weapon," and "location" using logic, questioning, and elimination techniques. The challenge is to create a game where both human and AI players can participate in a structured, turn-based environment to deduce the solution.

The game objectives include:

1. Allowing human and AI players to question each other about cards.
2. Implementing an AI that uses forward chaining and knowledge-based deduction to narrow down the possible solutions.
3. Enabling the player to win by correctly guessing the solution before the AI deduces it.

Game Setup:

Categories:

- **Suspects**: Miss Scarlet, Colonel Mustard, Mrs. White, Mr. Green, Mrs. Peacock, Professor Plum.
- **Weapons**: Knife, Candlestick, Revolver, Rope, Lead Pipe.
- **Locations**: Hall, Lounge, Kitchen, Ballroom, Conservatory, Attic.

Solution Cards:

- The game randomly selects one card from each category to form the solution (the true combination).
- These cards are hidden from both players.

Card Distribution:

- After selecting the solution, the remaining cards are shuffled and divided between the player and the AI.
- Each player, including the AI, receives half of the remaining cards, ensuring they hold some information about cards that are not in the solution.

2) SOFTWARE REQUIREMENTS:

a) Functional Requirements

Python Version:

- The game is written in Python and requires a compatible Python version.
- **Python 3.7 or higher** is recommended, with versions Python 3.8 and 3.9 offering compatibility with the latest libraries.

Required Libraries/Packages:

- **Tkinter**: Used for creating the game's graphical user interface, allowing interactive play with dropdowns and buttons for selection.
- **random**: A standard Python library used to handle random selection of solution cards.
- **messagebox**: A component of Tkinter, used for displaying game messages and announcements.

Installation:

- Tkinter is typically pre-installed with Python. If not available, install it via sudo apt-get install python3-tk on Linux systems or include Tkinter during the initial Python installation on Windows and macOS.

Development Environment (IDE):

- **VS Code**: A lightweight, customizable code editor with strong Python support.
- **IDLE**: Python's built-in IDE, which supports Tkinter applications and allows basic testing and debugging.

b) Non Functional Requirements:

1. **Performance**:
    - **Response Time**: The system should provide prompt responses within 1–2 seconds for each turn (for both human and AI).
    - **AI Decision Speed**: The AI should make deduction decisions within 2 seconds for standard difficulty, ensuring a seamless player experience.
2. **Scalability**:
    - The system should be adaptable to support future features, such as multiplayer mode, advanced AI reasoning, or additional categories of cards, without requiring major rework in the architecture.
3. **Usability**:
    - **Ease of Use**: The GUI interface (implemented using Tkinter) should be intuitive, allowing users to easily navigate, select cards, submit questions, and make guesses.
    - **User Assistance**: Instructions should be readily available within the game to guide new players on game rules and mechanics.
    - **Accessibility**: The interface should be accessible with simple, readable fonts and colors, designed to be visually engaging but also suitable for all users.
4. **Reliability**:
    - **Error Handling**: The game should manage invalid inputs (e.g., selecting cards that aren't available) gracefully, displaying error messages as needed without

crashing.
    - **System Stability**: The game should run without errors, crashes, or disruptions, ensuring game state is tracked accurately from start to finish.
5. **Maintainability**:
    - **Modularity**: The code should follow a modular structure, with clear functions ,making it easy to update, expand, or debug specific features without affecting the entire system.
    - **Documentation**: Code should include inline comments explaining the purpose of key sections, and external documentation should outline the game setup, flow, and AI logic.
    - **Code Reusability**: Functions, especially those managing AI deductions and player interactions, should be designed for potential reuse in similar games or game extensions.
6. **Portability**:
    - The game should be easily portable across different operating systems (e.g., Windows, macOS, Linux) and Python environments, without relying on platform-specific libraries or features.

3) DESIGN ALTERNATIVES:

1. **User Interface Alternatives**:
    - **Graphical User Interface (GUI)**:
        - **Tkinter (Current Approach)**: Tkinter, a built-in Python library, is used for the game's GUI, allowing players to select cards and interact via buttons. While effective, Tkinter could be expanded for additional visual feedback and improved layout.
        - **Pygame**: Using Pygame could enhance the game with custom animations, sound effects, and visual cues that highlight AI deductions or player choices, adding interactivity.
        - **Web-based GUI**: A web-based version using HTML, CSS, and JavaScript (with Python backend using Flask or Django) could make the game accessible in a browser, enabling more intuitive interactions through clicks and taps on mobile devices.
    -
    - **Text-based Interface**:
        - A Command-Line Interface (CLI) could serve as a lightweight, text-based version of the game, where players type in choices and receive feedback via text prompts. This approach reduces graphical overhead but is less engaging.

2. **AI Deduction Algorithm Alternatives**:
    - **Rule-Based Deduction (Current Approach)**:
        - The current AI uses a rule-based approach, leveraging forward chaining to eliminate possibilities based on the player’s responses. This approach is suitable for deduction games, but it may be improved with more complex

reasoning techniques.

- **Alternative AI Deduction Techniques**:
  - **Backward Chaining**: Instead of forward chaining, the AI could work backward from a hypothesized solution, asking questions to confirm or reject possibilities. This approach could make the AI feel more strategic and methodical.
  - **Machine Learning (ML)**: For a more advanced AI, ML models could be trained to recognize common deduction patterns and optimal questioning strategies. Using reinforcement learning, the AI could learn from game simulations to improve its questioning.
  - **Heuristic-Based Deduction**: A heuristic-based AI could prioritize questioning based on the number of remaining unknowns or the likelihood of each category based on earlier responses. This would add flexibility and efficiency to the AI's reasoning process.

3. **Performance Optimization Alternatives**:
   - **Parallel Processing for AI Deduction**:
     - For games involving complex deduction, the AI's question selection could be parallelized using Python's multiprocessing library, improving response times for AI turns by running deduction processes on multiple cores.

   - **Optimized Deduction Order**:
     - By optimizing the order of questions based on known information or high-probability guesses, the AI can reduce unnecessary questions. This "question ordering" technique helps the AI reach conclusions faster and minimizes redundant questions, making for smoother gameplay.

4. **Game Features Alternatives**:
   - **Multiplayer Mode (Local or Online)**:
     - **Local Multiplayer**: Allowing two players to take turns guessing could make the game more engaging and social. Players could compete to deduce the correct solution faster than their opponent.
     - **Online Multiplayer**: An online version where players can compete in real-time across devices could expand the game’s reach. Technologies like WebSockets or Flask-SocketIO would support real-time interactions, keeping both players in sync.

5. **Testing and Validation Alternatives**:
   - **Unit Testing**:
     - Using a framework like unittest or pytest would help verify game logic and AI deductions, ensuring functions handle edge cases correctly. For example, tests could validate AI responses to known cards, check deduction logic, and ensure the solution is deduced without errors.
   - **Automated Game Simulation**:
     - To evaluate the AI's performance, an automated simulation could pit two

AI agents against each other, testing the consistency and efficiency of deduction logic over multiple games. This would allow fine-tuning AI strategies and help measure AI performance across different scenarios.

These design alternatives provide various avenues for enhancing the game's user experience, AI sophistication, and performance, ensuring a more engaging, challenging, and seamless deduction game for players.

4) KEY FEATURES:

a)Solution and Card Selection:

1. At the start of the game, the program randomly selects one card from each category (suspects, weapons, and locations) to form the hidden solution.
2. The remaining cards are then shuffled and distributed equally between the AI and the player, providing each with clues about cards that are not part of the solution.

b) Player Turns and Questioning:

3. **Human Player**: The player takes turns selecting two cards from any category to question the AI. If the AI has one of these cards, it reveals it; otherwise, it responds with "No." This information allows the player to progressively eliminate possibilities.
4. **AI Player**: The AI has a similar process of elimination. It uses forward chaining and a knowledge base to track revealed cards and narrow down the solution by systematically questioning the player about their cards.

c) AI Deduction Logic:

5. The AI uses a knowledge-based deduction system. Through forward chaining, it progressively narrows down options by eliminating cards revealed by the player and updating its knowledge base.
6. **Deduction Functions**:
    - **ai_turn()**: The AI selects cards to ask about based on its remaining unknowns.
    - **ai_update_knowledge()**: Updates the AI's knowledge whenever it learns about a specific card, removing it from possible suspects, weapons, or locations.
    - **ai_check_solution()**: Determines if the AI has enough information to deduce the final solution.
    - **ai_make_guess()**: If the AI deduces all components of the solution, it can make a final guess to win the game.

d) Winning Condition:

7. A player wins if they correctly deduce the solution (murderer, weapon, and location).
8. The game checks for a winning guess after each turn, and if the player or AI accurately identifies the solution, they are declared the winner.

e) Game Termination:

9. The game concludes when:
    - **A player (human or AI) guesses correctly**, identifying the solution accurately and winning the game.
    - **The player or AI makes an incorrect guess**, allowing the game to continue until a correct guess is made.

These features create an engaging deduction game where logical reasoning, questioning, and elimination allow both the player and AI to work toward uncovering the hidden solution. The AI's systematic approach provides a challenging opponent for players to compete against.

5) ALGORITHM:

Forward Chaining Algorithm for AI Deduction

Forward chaining is used in this game to enable the AI to make deductions by gradually building knowledge about possible suspects, weapons, and locations. By systematically eliminating cards based on player responses, forward chaining helps the AI narrow down potential solutions in a logical sequence. This approach supports a step-by-step deduction process, allowing the AI to reach the correct answer by confirming clues as it gathers more information.

Given:

- Set of all suspects S, weapons W, and locations L.
- Solution cards are unknown to the AI: solution_murderer, solution_weapon, solution_location.
- AI's knowledge base:
    - ai_knowledge_suspects: suspects that AI knows are not the solution.
    - ai_knowledge_weapons: weapons that AI knows are not the solution.
    - ai_knowledge_locations: locations that AI knows are not the solution.
- AI's hand of cards ai_cards.

Algorithm:

1. **Initialize Knowledge Base:**
    - ai_knowledge_suspects ← S−ai_cards
    - ai_knowledge_weapons ← W−ai_cards
    - ai_knowledge_locations ← L−ai_cards
    - known_murderer ← None
    - known_weapon ← None
    - known_location ← None

2. **AI Turn: Asking Questions**
    - **Input:**
        - ai_knowledge_suspects
        - ai_knowledge_weapons
        - ai_knowledge_locations
    - **Output:** List of cards to ask (cards_to_ask)

- **Procedure:**
    i. If known_murderer is None:
        1. Select any suspect s from ai_knowledge_suspects not in ai_cards.
        2. Add s to cards_to_ask.
    ii. If known_weapon is None:
        1. Select any weapon w from ai_knowledge_weapons not in ai_cards.
        2. Add w to cards_to_ask.
    iii. If known_location is None:
        1. Select any location l from ai_knowledge_locations not in ai_cards.
        2. Add l to cards_to_ask.
- **Return** the first two cards in cards_to_ask.

3. **AI Deduction Based on "No" Response**
    - **Input:** question (cards asked about), player_response
    - **Output:** Updated ai_knowledge or known cards
    - **Procedure:**
        i. For each card ccc in question:
            1. If player_response is "No":
                a. If $c \in S$: Set known_murderer $\leftarrow$ c
                b. If $c \in W$: Set known_weapon $\leftarrow$ c
                c. If $c \in L$: Set known_location $\leftarrow$ c

4. **Check if Solution is Deduced**
    - **Input:** known_murderer, known_weapon, known_location
    - **Output:** Boolean is_solution_deduced
    - **Procedure:**
        1. Return True if known_murderer, known_weapon, and known_location are all set (not None), else False.

5. **AI Final Guess**
    - **Input:** known_murderer, known_weapon, known_location
    - **Output:** AI's final guess (murderer, weapon, location)
    - **Procedure:**
        1. Return {"murderer": known_murderer, "weapon": known_weapon, "location": known_location}

6)IMPLEMENTATION:

```
import tkinter as tk
from tkinter import messagebox
import random

# Define the possible cards
suspects = ["Miss Scarlet", "Colonel Mustard", "Mrs. White", "Mr. Green", "Mrs.
Peacock", "Professor Plum"]
weapons = ["Knife", "Candlestick", "Revolver", "Rope", "Lead Pipe"]
locations = ["Hall", "Lounge", "Kitchen", "Ballroom", "Conservatory","Attic"]

# Randomly choose the solution cards
solution = {
    "murderer": random.choice(suspects),
    "weapon": random.choice(weapons),
    "location": random.choice(locations)
}
print("For Reference: ")
print("Solution is: ",solution)
# Remove solution cards from the deck
remaining_suspects = [s for s in suspects if s != solution["murderer"]]
remaining_weapons = [w for w in weapons if w != solution["weapon"]]
remaining_locations = [l for l in locations if l != solution["location"]]

# Create a deck with remaining cards
deck = remaining_suspects + remaining_weapons + remaining_locations
random.shuffle(deck)

# Distribute cards to AI and player
ai_cards = deck[:len(deck) // 2]
player_cards = deck[len(deck) // 2:]

# AI knowledge base
ai_knowledge = {
    "suspects": [s for s in suspects if s not in ai_cards],
    "weapons": [w for w in weapons if w not in ai_cards],
    "locations": [l for l in locations if l not in ai_cards],
    "known_murderer": None,
    "known_weapon": None,
    "known_location": None
}

# Initialize tkinter root window
root = tk.Tk()
root.title("Cluedo Deduction Game")
root.geometry("1000x880")
root.configure(bg="lightblue")

# Display text area for game output
```

```
display_text = tk.Text(root, height=18, width=70, bg="lightyellow", wrap=tk.WORD,
font=("Helvetica", 14))
display_text.pack(pady=10)

# Insert welcome message and instructions
welcome_message = """
Welcome to Cluedo!

Instructions:
1. Try to deduce the correct combination of murderer, weapon, and location.
2. Each turn, ask the AI about specific suspects, weapons, or locations.
3. The AI will reveal a card if it has one, or respond with "No."
4. Use responses to narrow down the possibilities and make an informed guess.

Good luck!
Select Your Cards:
"""

# Display the message in the text area
display_text.insert(tk.END, welcome_message)


# Show all available cards
all_cards_label = tk.Label(root, text="All Available Cards:", bg="lightblue",
font=("Helvetica", 16, "bold"))
all_cards_label.pack()

# Display the suspects, weapons, and locations
suspects_label = tk.Label(root, text=f"Suspects: {', '.join(suspects)}",
bg="lightblue", font=("Helvetica", 14))
suspects_label.pack()
weapons_label = tk.Label(root, text=f"Weapons: {', '.join(weapons)}",
bg="lightblue", font=("Helvetica", 14))
weapons_label.pack()
locations_label = tk.Label(root, text=f"Locations: {', '.join(locations)}",
bg="lightblue", font=("Helvetica", 14))
locations_label.pack()

# Player's cards display label
player_cards_label = tk.Label(root, text=f"Your Cards: {', '.join(player_cards)}",
bg="lightblue", font=("Helvetica", 14, "bold"))
player_cards_label.pack(pady=10)


# Display the AI's knowledge base for reference
def print_knowledge_base():
    print("AI Knowledge Base:")
    print(f"Known Murderer: {ai_knowledge['known_murderer']}")
    print(f"Known Weapon: {ai_knowledge['known_weapon']}")
    print(f"Known Location: {ai_knowledge['known_location']}")
```

```
    print(f"Remaining Suspects: {ai_knowledge['suspects']}")
    print(f"Remaining Weapons: {ai_knowledge['weapons']}")
    print(f"Remaining Locations: {ai_knowledge['locations']}")
    print("\n")

# Update AI's knowledge base when a card is revealed
def ai_update_knowledge(revealed_card):
    if revealed_card in ai_knowledge["suspects"]:
        ai_knowledge["suspects"].remove(revealed_card)
    if revealed_card in ai_knowledge["weapons"]:
        ai_knowledge["weapons"].remove(revealed_card)
    if revealed_card in ai_knowledge["locations"]:
        ai_knowledge["locations"].remove(revealed_card)
    print_knowledge_base()

# AI deduces based on a "No" response from the player
def ai_deduce(question):
    for card in question:
        if card not in ai_cards:
            if card in suspects:
                ai_knowledge["known_murderer"] = card
            elif card in weapons:
                ai_knowledge["known_weapon"] = card
            elif card in locations:
                ai_knowledge["known_location"] = card
    print_knowledge_base()

# AI's turn using forward chaining
def ai_turn():
    cards_to_ask = []
    if not ai_knowledge["known_murderer"]:
        remaining_suspects = [s for s in ai_knowledge["suspects"] if s not in
ai_cards]
        if remaining_suspects:
            cards_to_ask.append(remaining_suspects[0])
    if not ai_knowledge["known_weapon"]:
        remaining_weapons = [w for w in ai_knowledge["weapons"] if w not in
ai_cards]
        if remaining_weapons:
            cards_to_ask.append(remaining_weapons[0])
    if not ai_knowledge["known_location"]:
        remaining_locations = [l for l in ai_knowledge["locations"] if l not in
ai_cards]
        if remaining_locations:
            cards_to_ask.append(remaining_locations[0])
    return cards_to_ask[:2]

# Check if AI can guess the solution
def ai_check_solution():
    return all([
```

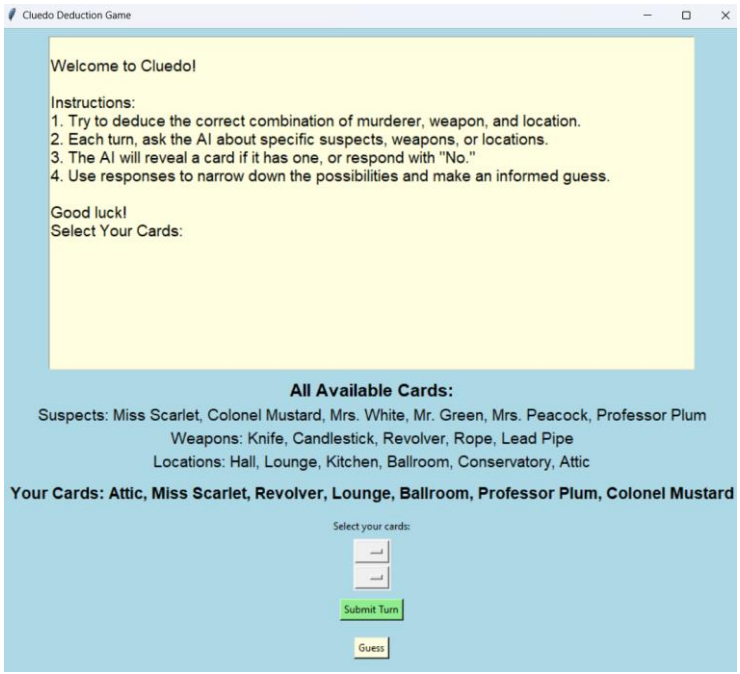
```
            ai_knowledge["known_murderer"],
            ai_knowledge["known_weapon"],
            ai_knowledge["known_location"]
        ])

    # AI makes a final guess
    def ai_make_guess():
        return {
            "murderer": ai_knowledge["known_murderer"],
            "weapon": ai_knowledge["known_weapon"],
            "location": ai_knowledge["known_location"]
        }

    # Handle a player's turn
    def player_turn():
        question = [player_card1.get(), player_card2.get()]
        response = handle_response(question, ai_cards)
        display_text.insert(tk.END, f"Player asked about: {question}\n")
        if response != "No":
            display_text.insert(tk.END, f"AI reveals: {response}\n")
        else:
            display_text.insert(tk.END, "AI responds: No\n")
        display_text.update_idletasks()

        # Enable the guess button after the player's turn
        guess_button.config(state=tk.NORMAL)

        # Automatically proceed to AI's turn
        ai_turn_main()

    # Handle response for AI and player
    def handle_response(question, opponent_cards):
        for card in question:
            if card in opponent_cards:
                return card
        return "No"

    # Main game loop for the AI turn
    def ai_turn_main():
        question = ai_turn()
        response = handle_response(question, player_cards)
        display_text.insert(tk.END, f"\nAI asks about: {question}\n")
        if response != "No":
            display_text.insert(tk.END, f"Player reveals: {response}\n")
            display_text.insert(tk.END, "\nSelect Your Cards:\n")
            ai_update_knowledge(response)
        else:
            display_text.insert(tk.END, "Player responds: No\n")
            display_text.insert(tk.END, "\nSelect Your Cards:\n")
            ai_deduce(question)
```

```
    if ai_check_solution():
        ai_guess = ai_make_guess()
        display_text.insert(tk.END, f"AI has deduced the solution!\nThe solution is
{ai_guess}\n")
        announce_winner("AI")
    display_text.update_idletasks()

# Function to announce the winner
def announce_winner(winner):
    if winner == "Player":
        messagebox.showinfo("Game Over", "\nCongratulations! You have deduced the
correct solution and won the game!")
    else:
        messagebox.showinfo("Game Over\n", f"The AI has won! The correct solution
was: {solution}")

# Create dropdown for player card selection
player_card1 = tk.StringVar()
player_card2 = tk.StringVar()

# Dropdown menus for player turn
def create_player_dropdowns():
    tk.Label(root, text="Select your cards:", bg="lightblue").pack(pady=5)
    tk.OptionMenu(root, player_card1, *suspects + weapons + locations).pack()
    tk.OptionMenu(root, player_card2, *suspects + weapons + locations).pack()

def submit_player_turn():
    player_turn()

def guess_solution():
    guess_window = tk.Toplevel(root)
    guess_window.title("Make Your Guess")

    tk.Label(guess_window, text="Guess the Murderer:").pack()
    murderer_guess = tk.StringVar()
    tk.OptionMenu(guess_window, murderer_guess, *suspects).pack()

    tk.Label(guess_window, text="Guess the Weapon:").pack()
    weapon_guess = tk.StringVar()
    tk.OptionMenu(guess_window, weapon_guess, *weapons).pack()

    tk.Label(guess_window, text="Guess the Location:").pack()
    location_guess = tk.StringVar()
    tk.OptionMenu(guess_window, location_guess, *locations).pack()

    def check_guess():
        if (murderer_guess.get() == solution["murderer"] and
                weapon_guess.get() == solution["weapon"] and
                location_guess.get() == solution["location"]):
            announce_winner("Player")
```

```
        else:
            messagebox.showinfo("Incorrect Guess\n", f"You guessed incorrectly. The
correct solution was: {solution}")
        guess_window.destroy()

    tk.Button(guess_window, text="Submit Guess", command=check_guess).pack(pady=10)

# Create player dropdowns
create_player_dropdowns()

# Labels and Buttons
tk.Button(root, text="Submit Turn", command=submit_player_turn,
bg="lightgreen").pack(pady=10)
guess_button = tk.Button(root, text="Guess", command=guess_solution,
bg="lightyellow")
guess_button.pack(pady=10)

root.mainloop()
```

7)OUTPUT AND TEST CASES:

Initial Screen:

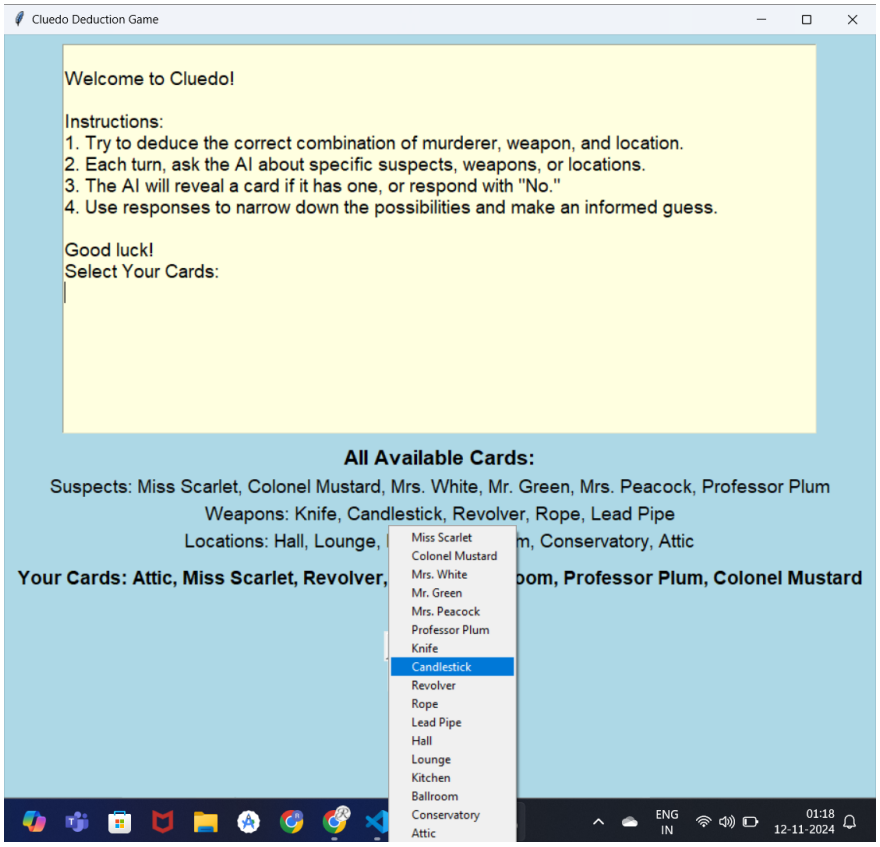


Initial Knowledge Base of AI:

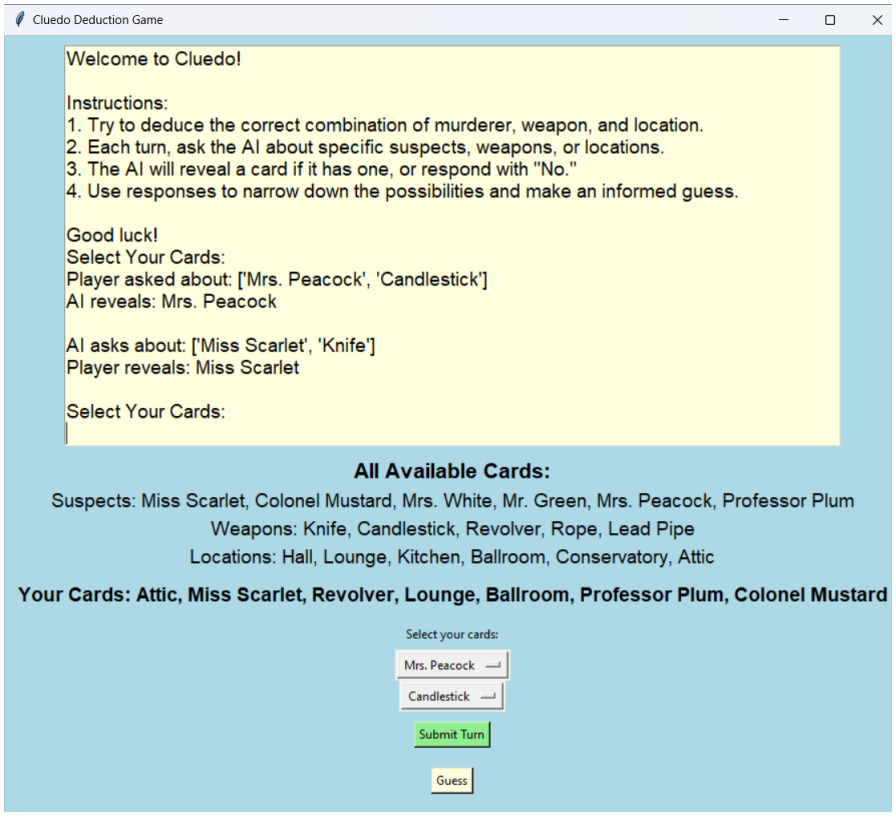
```
AI Knowledge Base:
Known Murderer: None
Known Weapon: None
Known Location: None
Remaining Suspects: ['Miss Scarlet', 'Colonel Mustard', 'Mr. Green', 'Professor Plum']
Remaining Weapons: ['Knife', 'Revolver']
Remaining Locations: ['Hall', 'Lounge', 'Ballroom', 'Attic']
```

Player can select the cards to ask AI:

The player can ask the AI whether it has Mrs. Peacock or Candlestick.
The player asks by pressing the Submit Turn button.



AI's Response and AI's Turn:



The AI has responded with Mrs Peacock, meaning that AI has Mrs. Peacock.
Next, AI asks the player whether the player has Miss Scarlet or Knife.
The player responds with Miss Scarlet.

Updated Knowledge Base of AI after Response of Player:

In remaining suspects, Miss Scarlet has been removed.

```
AI Knowledge Base:
Known Murderer: None
Known Weapon: None
Known Location: None
Remaining Suspects: ['Colonel Mustard', 'Mr. Green', 'Professor Plum']
Remaining Weapons: ['Knife', 'Revolver']
Remaining Locations: ['Hall', 'Lounge', 'Ballroom', 'Attic']
```

Player detection and AI detection:

a.Player:

Player has asked Mr. Green and Colonel Mustard.

AI responds with No.

From this we can infer that Mr Green is the suspect person as we already haveColonel Mustard.

b.AI

AI asked Mr. Green and Knife

Player has responded with NO

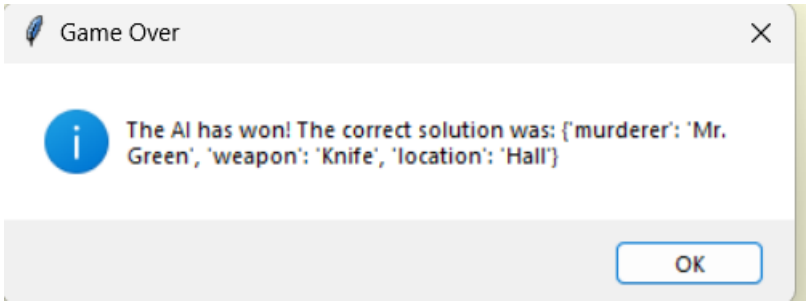
From this, AI (using forward chaining) finds out that the suspect person and suspect weapon are Mr. Green and Knife.

Updated Knowledge Base after No Response:

```
AI Knowledge Base:
Known Murderer: Mr. Green
Known Weapon: Knife
Known Location: None
Remaining Suspects: ['Mr. Green', 'Professor Plum']
Remaining Weapons: ['Knife', 'Revolver']
Remaining Locations: ['Hall', 'Lounge', 'Ballroom', 'Attic']
```

AI Finds the Answer:

AI makes a guess when murderer, weapon and place have been confirmed.

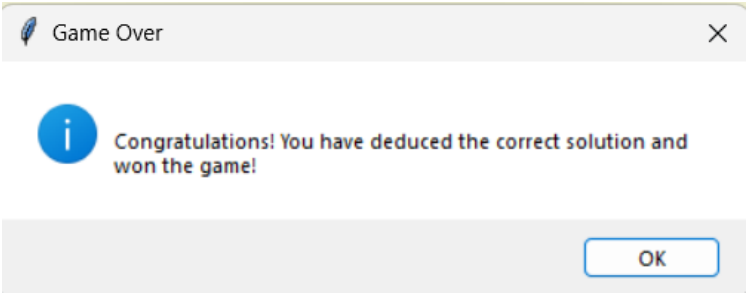



```
AI Knowledge Base:
Known Murderer: Mr. Green
Known Weapon: Knife
Known Location: Hall
Remaining Suspects: ['Mr. Green', 'Professor Plum']
Remaining Weapons: ['Knife', 'Revolver']
Remaining Locations: ['Hall', 'Lounge', 'Ballroom', 'Attic']
```

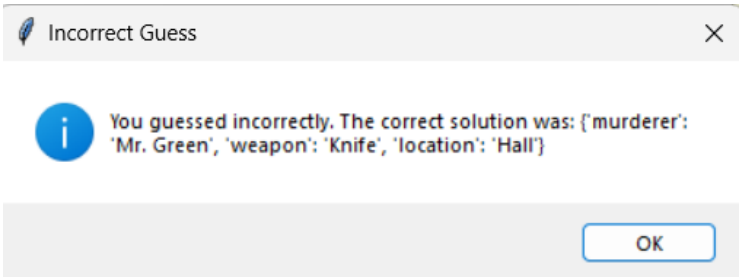
Player Makes a guess:

If a player wants to guess the answer, he/she does that by pressing the guess button. Makes his/her guess and presses the Submit button.

If correct, player wins!



Else, AI wins



8) TECHNOLOGICAL IMPROVEMENTS:

1) **Enhanced AI Logic**: Implement additional heuristics or minimax-based decision-making for more advanced AI deduction.
2) **Advanced GUI Features**: Add animations and sound effects to make the game more engaging.

3) Multiplayer Support: Enable two players to play against each other locally or over a network.

4) **Save and Replay Options**: Allow the player to save progress and revisit previous games.

9) CONCLUSION:

This Cluedo Game assignment successfully demonstrates the integration of knowledge-based AI with forward chaining to create an engaging, logical deduction game. By implementing a turn-based questioning mechanism, both the player and AI can systematically narrow down possible solutions. The AI's use of forward chaining enables it to make informed guesses, providing a challenging opponent for the player. The game's structure, including GUI elements and logical deduction features, makes it both functional and enjoyable. With potential improvements such as advanced AI logic, multiplayer options, and save features, this game can evolve into an even more immersive and strategic experience. Through this assignment, key concepts in AI reasoning and game mechanics were explored, enhancing our understanding of AI-driven problem-solving in interactive environments.