# Project 2 Part 2 Cluedo Game With AI Players

**Name: Benyamin Plaksienko**

**NJIT UCID: bp446**

**Email Address: bp446@njit.edu**

**Professor: Dr. Arashdeep Kaur**

**Course: CS 670 Artificial Intelligence**

## Abstract

I created a working version of Cluedo just for this project, focusing on the basic game setup and mechanics as well as the design of a logic driven AI player. The system incorporates turn based movement, suggestion/refutation handling, accusation logic, deterministic solution selection, card dealing (with optional seeding), and a representation of characters, weapons, and chambers. The AI player updates its conclusions about the solution and opponents' hands, maintains an internal knowledge base, models belief distributions over the secret solution, and takes into account evidence from proposals, including ones that haven't been challenged. It selects movement and suggestions to optimize information gain, evaluates uncertainty using metrics like entropy, and only accuses when it has a high enough level of confidence in the solution. This project demonstrates practical application of AI techniques knowledge representation, inference, and decision making within an interactive game environment.

## Introduction

As part of my Artificial Intelligence course, I created this Cluedo game to show how AI methods may be used in a structured game that relies on deduction. Cluedo is a perfect choice for an AI focused project since it offers a rich environment for practicing knowledge representation, logical inference, and decision making under uncertainty. In addition to incorporating the game's fundamental elements movement, suggestions, refutations, and accusations I was driven to include an AI player who could actively engage by applying reasoning techniques akin to those a human player might use. With this project, I was able to incorporate course concepts such as **constraint satisfaction, probability based reasoning, and inference rules** into a functional gaming environment that demonstrates the usefulness of AI in strategic play and problem solving. I also implemented **Limited Uniform Cost Search** from the book to list rooms players can move to. I stored the mansion in a dictionary which I translated to a json file. I also stored the hands every player has and the solution in a dictionary which I translated to a json file. I implemented secret rooms that cost nothing to enter, and made it possible using random seed values to generate new mansion layouts and new solutions/hands.

## Game Rules

In my implementation, the Cluedo game proceeds with human and AI players taking turns to gather information and deduce the hidden solution. The rules and mechanics as coded are:

1. Mansion Setup
    - I generate the mansion layout by loading rooms and constructing a weighted graph with optional secret passages. The graph (including secret passage connections) is saved and reloaded to drive movement.
2. Solution and Dealing
    - One character, one weapon, and one room are randomly selected (with a seed for reproducibility) to form the hidden murder solution. The remaining cards are shuffled and dealt round robin to the active players (configurable number of total and AI players). Hands are stored per player.

3. Player Configuration and Initialization

    - The user specifies total players and how many are AI. All players are initialized at a randomly chosen starting room (same for everyone, based on seed), and their location history is tracked. AI players are instantiated with their own knowledge-tracking logic; human players receive interactive prompts.

4. Turn Structure

    - Players take turns in a fixed order. On a turn, a player may first choose to make an accusation.

    - If no immediate accusation is made or if the player isn't eliminated, movement occurs: the player "rolls" a six sided die, and their reachable rooms are computed via a limited uniform cost search on the mansion graph constrained by the die roll cost.

    - The player selects a valid room to move to (AI chooses based on its internal heuristics, humans input directly), and the move updates their location and history.

5. Suggestions

    - Upon entering a room, the player makes a suggestion consisting of a suspect (character), a weapon, and the room they are currently in.

- If the suggested character is elsewhere, their token is immediately moved into the suggested room (as per classic rules), and their history updated.
- Other players are queried in clockwise/turn order (starting with the next player) to see if they can refute the suggestion. If a player has one or more of the suggested cards, they privately show exactly one to the suggester; the process stops at the first refuter.
- If no one refutes, the suggestion is recorded as unrefuted, which the AI uses to update belief about the solution.

6. Accusations
    - A player may make an accusation before or after movement/suggestion on their turn. An accusation names a full solution (character, weapon, room).
    - If the accusation matches the hidden solution, that player wins immediately and the game ends.
    - If incorrect, the accuser is marked inactive for future movement, suggestions, or accusations but remains in the game as a refuter for others' suggestions.
7. AI Behavior
    - AI players maintain an internal knowledge base: their own hand, seen cards, observed refutations, and unrefuted suggestions. They maintain belief distributions over possible solution cards and update these based on evidence (shown cards and unrefuted

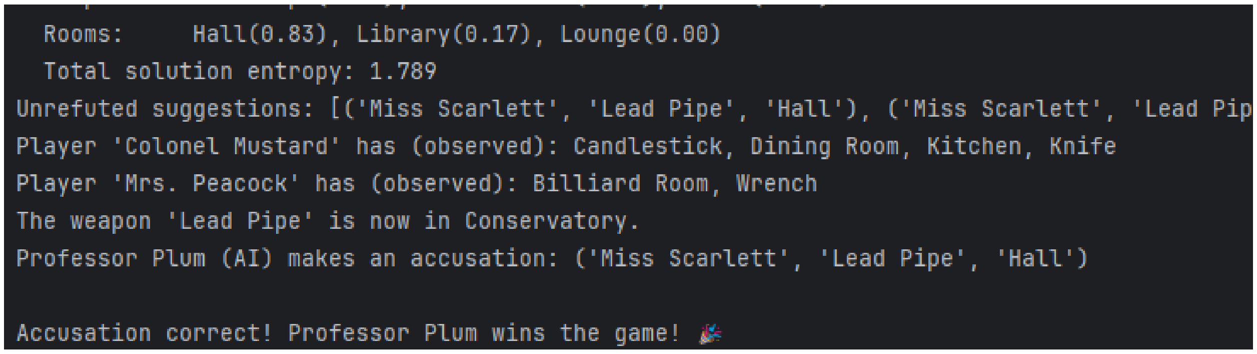
suggestions).

- The AI uses this knowledge to choose movement, make suggestions, and determine whether to accuse. It only makes an accusation when its confidence (via its internal threshold logic) in the solution is high enough.
- All suggestions (whether made by the AI or observed from others) are fed into each AI’s inference routines for future reasoning.

8. Weapon Tracking
    - When a suggestion includes a weapon, that weapon’s current location is updated to the room where the suggestion was made.
9. Win/Loss Conditions
    - Correct accusation: immediate victory.
    - Incorrect accusation: player loses ability to act offensively but can still contribute defensively by refuting suggestions.
    - If all active players are eliminated via incorrect accusations without anyone correctly solving the case, the game ends with no winner.

**Thorough Testing**

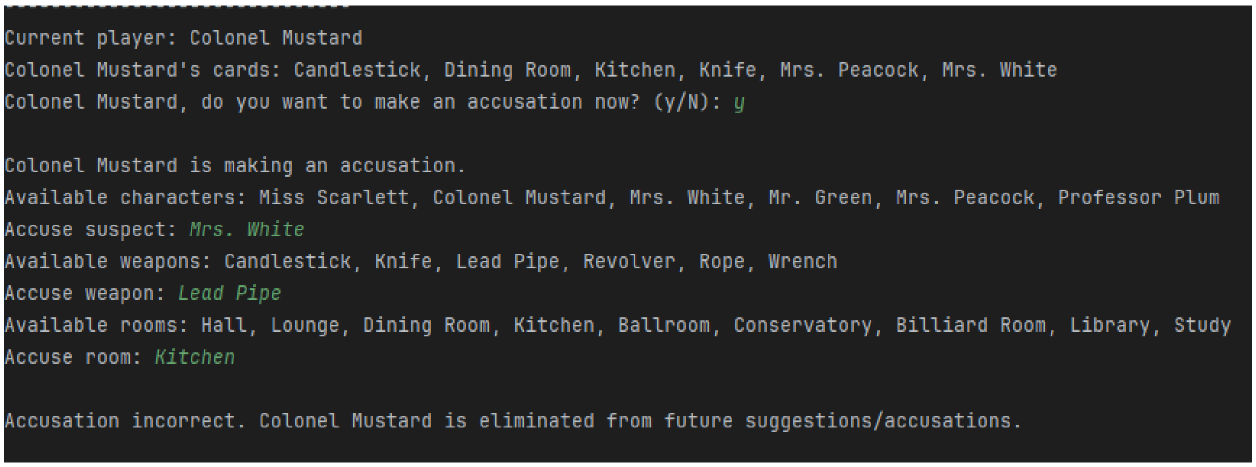
- **Successful game completion (correct accusation).**



```
  Rooms:     Hall(0.83), Library(0.17), Lounge(0.00)
  Total solution entropy: 1.789
Unrefuted suggestions: [('Miss Scarlett', 'Lead Pipe', 'Hall'), ('Miss Scarlett', 'Lead Pip
Player 'Colonel Mustard' has (observed): Candlestick, Dining Room, Kitchen, Knife
Player 'Mrs. Peacock' has (observed): Billiard Room, Wrench
The weapon 'Lead Pipe' is now in Conservatory.
Professor Plum (AI) makes an accusation: ('Miss Scarlett', 'Lead Pipe', 'Hall')

Accusation correct! Professor Plum wins the game! 🎉
```

- **Incorrect accusation (player elimination).**



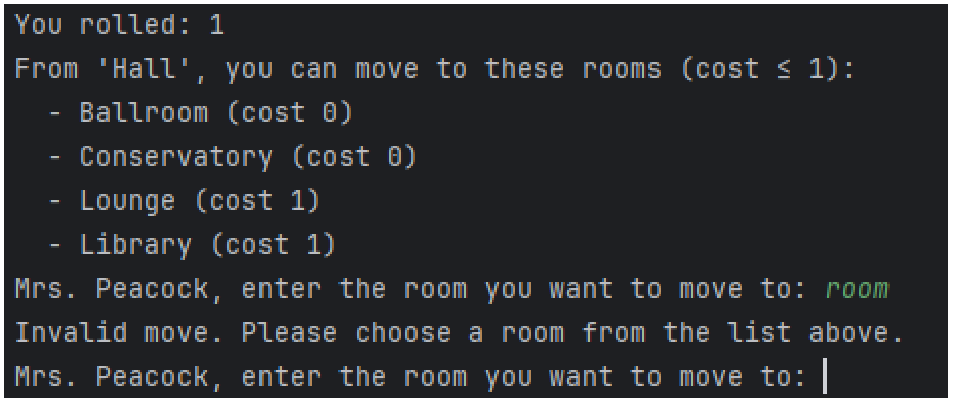
```
Current player: Colonel Mustard
Colonel Mustard's cards: Candlestick, Dining Room, Kitchen, Knife, Mrs. Peacock, Mrs. White
Colonel Mustard, do you want to make an accusation now? (y/N): y

Colonel Mustard is making an accusation.
Available characters: Miss Scarlett, Colonel Mustard, Mrs. White, Mr. Green, Mrs. Peacock, Professor Plum
Accuse suspect: Mrs. White
Available weapons: Candlestick, Knife, Lead Pipe, Revolver, Rope, Wrench
Accuse weapon: Lead Pipe
Available rooms: Hall, Lounge, Dining Room, Kitchen, Ballroom, Conservatory, Billiard Room, Library, Study
Accuse room: Kitchen

Accusation incorrect. Colonel Mustard is eliminated from future suggestions/accusations.
```

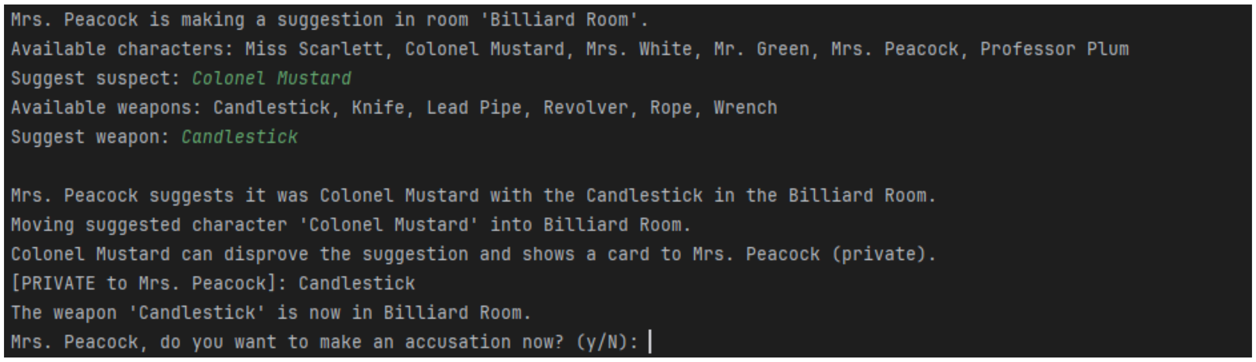
- **Movement tests (valid and invalid moves).**

  In my code, it isn't possible to make an invalid move because it lists all the valid moves, and if the chosen move isn't one of them, it displays an error and prompts the player to enter a valid move again.



```
You rolled: 1
From 'Hall', you can move to these rooms (cost ≤ 1):
  - Ballroom (cost 0)
  - Conservatory (cost 0)
  - Lounge (cost 1)
  - Library (cost 1)
Mrs. Peacock, enter the room you want to move to: room
Invalid move. Please choose a room from the list above.
Mrs. Peacock, enter the room you want to move to: |
```

- **Suggestion and refutation scenarios (e.g., all cards refuted, no cards refuted, specific card shown).**



```
Mrs. Peacock is making a suggestion in room 'Billiard Room'.
Available characters: Miss Scarlett, Colonel Mustard, Mrs. White, Mr. Green, Mrs. Peacock, Professor Plum
Suggest suspect: Colonel Mustard
Available weapons: Candlestick, Knife, Lead Pipe, Revolver, Rope, Wrench
Suggest weapon: Candlestick

Mrs. Peacock suggests it was Colonel Mustard with the Candlestick in the Billiard Room.
Moving suggested character 'Colonel Mustard' into Billiard Room.
Colonel Mustard can disprove the suggestion and shows a card to Mrs. Peacock (private).
[PRIVATE to Mrs. Peacock]: Candlestick
The weapon 'Candlestick' is now in Billiard Room.
Mrs. Peacock, do you want to make an accusation now? (y/N): |
```

- **Edge cases you considered (e.g., last player remaining, only one card to refute).**

  If there is only one player remaining he goes until he makes wrong accusation or correct one

- **Demonstrate how your implemented AI player makes deductions or strategic choices through relevant screenshots or log outputs.**

  In my implementation, the AI player makes deductions and strategic choices by keeping an internal knowledge base that is organized using Python dictionaries, sets, and lists. I use a nested dictionary called p_solution to track the AI's belief probabilities for every suspect, weapon, and room being part of the hidden solution, with the card names as keys and floating point probabilities as values. This lets the AI adjust its reasoning in real time as new information is discovered. I store the AI’s own cards and all the cards it has seen in sets so I can quickly check if a card is already known and efficiently remove it from possible solution candidates. I also use dictionaries like player_has and player_cannot_have to keep track of which cards other players are confirmed to have or cannot have, and these each contain sets for the three categories (character, weapon, and room). For ordered information, like the AI’s movement history or a list of unrefuted suggestions, I use lists so I can preserve the sequence of events. During gameplay, if another player shows a card, the AI updates its p_solution dictionary to set that card’s probability to zero and then normalizes the rest. If a suggestion goes unrefuted, the AI increases the probability for each of those suggested cards. These probability updates reduce uncertainty, which I measure using entropy, and they directly influence the AI’s decisions. The AI will move toward rooms that have higher probability in its dictionary or ones it has not visited recently, and it will make suggestions that are most likely to reveal new information. Finally, it will only make an accusation when the probability for a specific suspect, weapon, and room is high enough (above a set threshold), ensuring it doesn’t guess too early. By combining these different data structures with probability based reasoning, my AI is able to think logically and adapt its strategy as the game progresses.

## Challenges Faced

The project's AI player component was my main concern. In its early iterations, my AI made random choices for movement and suggestions rather than using any probabilities at all. Although this made it possible for the game to function, it also rendered the AI's actions irrational and ineffectual, frequently recommending the same cards over and over again or making irrational accusations. I soon saw that this method didn't highlight the abilities I intended to display in terms of thinking and inference. Another problem I faced was figuring out how to store and update the AI's knowledge as the game progressed. My first attempt was to keep all information in one big list, but this quickly became messy and made it hard to check if a card had been seen before or ruled out. This caused the AI to make strange choices, like suggesting cards it already knew weren't part of the solution. To fix this, I redesigned the AI's data structures to use nested dictionaries for probability tracking, sets for cards it had seen or owned, and lists for maintaining an ordered history of events such as unrefuted suggestions. I also implemented a probability based threshold so the AI would only make an accusation when it had strong evidence, instead of guessing too early. One of the trickier parts was refining the refutation logic so the AI could learn from other players' interactions without making false eliminations. Debugging this involved printing the AI's belief state after every turn and carefully tracing how probabilities changed. In the end, moving from a random selection AI to a probability driven, knowledge based AI not only made the game more competitive but also gave me valuable experience in applying AI reasoning techniques to a complex problem.

**Stability and Reliability**

My code is very stable, because it doesn't allow wrong inputs and asks to make a new input if the input is completely wrong.

**Additional Details**

How to run->

Python game.py

Environment->

I used Python 3.11 with Pycharms virtual environment it generates.

Future renovations->

If I had more time I would probably develop a more engaging front-end or web application, I would also make a back-end to store important information such as users, and make the game playable between multiple devices.

**AI/Logic Component Deep Dive**

The AI player in my Cluedo game uses probability based reasoning and knowledge tracking to make smart decisions. It maintains a belief system over all possible solution cards (characters, weapons, and rooms) and updates these beliefs as the game progresses.

To represent what it knows, the AI uses these main data structures:

- Probability dictionaries (p_solution) track the likelihood that each character, weapon, or room is part of the actual murder solution. Initially, all cards start with equal probabilities, except the AI's own hand cards, which are set to zero because they can't be part of the solution. The AI continuously normalizes these probabilities to make sure they sum to 1 within each category.

- Sets store cards that the AI owns and cards it has seen revealed by other players, helping it exclude those cards from suspicion.

- For modeling other players, the AI keeps two dictionaries:

    - player_has stores cards that players have definitely shown,

    - player_cannot_have stores cards players are known not to have.

- A list of unrefuted suggestions records suggestions that no one disproved, which increases the AI's belief that those cards might be in the solution.

The AI updates its knowledge when:

- It sees a card shown by another player. The AI sets that card's solution probability to zero and records that the player has it.

- A suggestion goes unrefuted, which causes the AI to increase probabilities of the suggested cards because no one could disprove them.

The AI uses an entropy function to measure uncertainty over the solution probabilities, aiming to reduce entropy by gathering better information.

For making suggestions, the AI picks characters and weapons with intermediate probabilities that maximize information gain usually the ones with the most uncertainty to learn more from other players' responses. It always suggests the room it currently occupies.

When choosing where to move, the AI scores reachable rooms based on their current solution probability (favoring rooms likely to be part of the solution) and whether it has visited them recently (favoring unexplored rooms).

The AI decides to make an accusation only when its confidence (probability) for one character, weapon, and room each exceeds 80%, minimizing premature or risky guesses.

Finally, the AI maintains a detailed knowledge summary, showing what it knows about its own cards, seen cards, other players' holdings, unrefuted suggestions, and the current probability distribution over solution cards. This summary can be printed for debugging or understanding the AI's reasoning process.

Overall, this combination of nested dictionaries, sets, lists, probability updates, and entropy calculations allows the AI to continuously refine its understanding and act strategically throughout the game, mimicking how a human player might think logically and adjust their guesses based on observed evidence.