Propositional Logic: Clue Game Reasoner

Due Nov 23, 2020 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** py

Goal

The goal of this assignment is to use propositional logic and a satisfiability reasoning tool to help us solve a deduction game. Ideally, this will give you insight into how to use propositional logic for problems that are bigger than "toy"-style logic problems. We won't build a complete logic-based agent, but what we do build here could easily be incorporated into an agent that would navigate the stochastic elements of the board game.

The Game of Clue

We are going to use propositional logic to represent information needed to help solve a game of **clue (https://en.wikipedia.org/wiki/Cluedo)** . (The game has had many versions of its rules over the years, and we will use the rules, rooms, characters and items from the 1960 through 1996 editions. Apologies if you learned a different edition.)

If you aren't familiar with the game, don't worry! It has some simple stochastic elements (that we will ignore) but the main basis of the game is deduction: Someone has committed a murder, and our job is to find out who committed it, in what room, and with which weapon.

In each game of clue, there is a deck of cards. On each card is either a suspect, a weapon or a room. There are six suspects, six weapons, and nine rooms for a total of 21 cards.

At the beginning of the game, one suspect, one weapon and one room is randomly selected to go into an envelope known as the "Case File". The remaining cards are dealt to the players.

As the game progresses, your goal is deduce which specific cards are in the Case File. Those cards represent the culprit, the murder weapon and the location that the murder occurred.

The way that you go about this deduction is by asking other players questions about what is in their hand. We will get into the specifics of this later. As you find out more information about what cards are where, you are able to eliminate some possibilities and eventually you should be able to solve the mystery! (Hopefully before anyone else!)

Propositional Logic Variables

We need to decide how to describe our world in the language of propositional logic. For this game, what we need to figure out is where cards are. They can be in the Case File or in a player's hand. We will be playing with 6 players, so we will need a boolean variable to represent each card being in any of the seven possible locations.

For example, one of the cards is the "Revolver". Therefore, we will need seven boolean variables, each of which represent that weapon being in either a player's hand or the Case File. This card must be exactly in one location (card locations don't change during the course of the game) so we can add the propositional logic sentences that enforce that exactly one of those seven variables is true.

Clue Reasoner Code

You will do your work in the clue_game_reasoner.py module that is provided.

In the code provided, the cards are represented with two-letter abbreviations (sorry.)

The suspects are: Miss Scarlet (sc), Colonel Mustard (mu), Miss White (wh), Mr. Green (gr), Miss Peacock (pe), Professor Plum (pl)

The weapons are: Knife (kn), Candlestick (ca), Revolver (re), Rope (ro), Pipe (pi), Wrench (wr)

The rooms are: Hall (ha), Lounge (lo), Dining Room (di), Kitchen (ki), Ballroom (ba), Conservatory (co), Billiards Room (bi), Library (li), Study (st)

Confusing our terminology is the fact that in the game, the players play *as* suspects. So there is a card named "Miss Scarlet" but also one of the players is playing as "Miss Scarlet". In the code provided, uppercase letters indicate players (and the case file) and lowercase letters to indicate cards.

Thus, when you create a boolean variable for our world, you will want to combine the abbreviation for the card and the abbreviation for the location. For example, when I wrote my solution, I used the string "kn_SC" for the variable representing the Knife card being in Miss Scarlet's hand.

The code is written as a class that is initialized with the ordering of the players in the game. The job of the constructor (the __init__ method) is to add all the clauses needed to represent the rules of game in propositional logic.

The job of the methods *add_hand*, *suggest*, and *accuse* is to handle things that happen during the game and they will be described later.

Your Task

Download the **clue_hw.zip** **(https://canvas.umn.edu/courses/193654/files/16917871/download?download_frd=1)** file and unzip it. You will do all your work in the clue_game_reasoner.py module.

Add code to the places where there is a comment "TO BE IMPLEMENTED AS AN EXERCISE".

Here are those places and what you need to add:

Constructor (the __init__ method)

Add to the Knowledge Base (KB) the logic that encodes the rules of the game:

- Each card is in at least one location (this is done for you as an example)
- Each card is in at most one location
- At least one card of each category (Suspects, Weapons, Rooms) is in the Case File
- At most one card of each category is in the Case File

add_hand

This method adds to the KB the information about the cards in your hand. (If they are in your hand, you know where they are.)

suggest

This method is the most complex and the interesting part of the game. A "suggestion" in Clue is when you say "I suggest that a particular character is guilty of the crime in a particular room with a particular weapon." According to the rules, when you make a suggestion, the other players will attempt to refute the suggestion. Starting with the player on your left, and going around the table until refuted, the other players must refute the suggestion if they can (iby revealing a card from their hand to the suggester) or pass if they cannot. Once a suggestion has been refuted, then no other player needs to refute it.

So what information is gained? Well, clearly if I made a suggestion that Professor Plum did it with the Candlestick in the Ballroom, and the player to my left (Colonel Mustard) shows me the Ballroom card, then I can add the knowledge that the Ballroom card is in Colonel Mustard's hand.

But what if Colonel Mustard passes and the next player, Mrs. White shows me the Ballroom? Certainly I can add that the Ballroom card is in Mrs. White's hand. But I can **also** add that none of the Candlestick, Ballroom or Professor Plum cards are in Colonel Mustard's hand.

Can I gain any information when someone **else** makes a suggestion? Yes!

If Colonel Mustard suggests a scenario, and then Miss White refutes it, then I know that Miss White has at least one of the three suggested cards in her hand.

If Colonel Mustard suggests a scenario and Miss White passes, and then Mr. Green refutes, then I know that Miss White has none of the three suggested cards, and that Mr. Green has at least one of the three suggested cards.

Keep in mind that a player is allowed to suggest cards that are in their own hand (a common strategy), so be careful not to make assumptions about that.

The suggest method takes parameters of who made the suggestion, the three cards involved, who refuted (which could be None) and what card was shown (which might be None if we don't know.)

Add code to the suggest method that will handle all of these scenarios, create clauses with the information gained, and add them to the KB.

The list *self.players* contains the turn order of the players in the game.

accuse

The last way that we can learn knowledge is when someone makes an accusation. (Technically, an accusation either ends the game or results in the accuser losing the game, but we will ignore this aspect and only focus on the information gained.)

An accusation is when a player tries to win the game by naming the three cards in the Case File. They name three cards and then look in the Case File. If they are correct, they win the game. If they are not, they can no longer win but must remain in the game to refute suggestions.

Add code to the *accuse* function that will create clauses based on information gained by a player's accusation and add them to our KB.

Testing

The code comes with three sample games. (Technically two separate games, but one of them is shown from the perspective of two different players.) We played games and tracked each suggestion.

The sample games call the *print_notepad* method near the end to show what information that we have. If you have implemented your methods correctly, the SAT Solver should be able to reason out enough information to answer the question of what exactly is in the Case File.

But Beware! SAT is an NP-Complete problem and we're solving it twice for each variable that we have in order to generate the notepad. So if you didn't add all the clauses correctly, it might take a really long time to generate the notepad. In my solution, it takes about 20 seconds to generate the notepad. But before I had all the clauses in there properly, it tended to get stuck on some of the notepad items. So when you're testing, you may want to rewrite the *print_notepad* method to focus on one row or column.

Submission

Submit your version of clue_game_reasoner.py via Canvas.

Some Rubric (1)

Criteria	Ratings		Pts
Constructor	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
add_hand	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
suggest	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
accuse	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
			Total Points: 20