Clue Detective

Reflection on W200 Project 1

Clue Detective is a smart note-taking program designed as a companion to playing the classic board game Clue (how to play). The game consists of five modules: `clue.py`, `board.py`, `detective.py`, `errors.py`, and `printouts.py`.

Looking Back

This was my first real coding project in any language, and my inexperience definitely showed. While my original design document captured the important features well, I needed to change my proposed implementation a lot while I was actually coding. For example, I did not anticipate the amount of error handling I would need to ensure a user experience that met my minimum standards. I also coded the entire project in Jupyter Notebook instead of an IDE because it was most familiar. While I was able to overcome this by extensively testing my code, I wasted a lot of time on simple bugs that an IDE could have caught easily. I'm definitely using an IDE in project 2!

I faced three main technical challenges in this project. The first is how to implement the matrix-like notebook that Clue players are so familiar with. I really wanted to be able to use a dataframe, but I would need to use something beyond the Python standard library to do so. Instead, I ended up using a dictionary that stores the "coordinates" of each note as a card and player tuple. However, this implementation came with the second technical challenge: since card and player are my own classes, I would need to make them hashable in order to use them as keys, but I was not familiar with how to do that. To get around this, I ended up using the string representations of class and player as keys instead. This segues into the third technical challenge, which is slightly related. In the game Clue, the players pick one of the suspects as their token for game play, but the players are actually independent from the suspects represented by the cards. This is reflected in my program in the fact that they are separate classes. However, because of this, comparing equivalent suspects become tricky: I need to constantly keep track of whether the suspect I'm talking about is a card, a player, or the string representation of either. Eventually, I was able to get around this by just always using string representations for comparisons of suspects, but not before I had to debug a lot of bewildering errors.

Looking Ahead

Overall, I am happy with my implementation of Clue Detective. I delivered on all the basic features I promised in my design document, and even managed to add some features like extensive error handling. My one regret, if you can call it that, is not being able to thoroughly optimize the way I wrote my functions. If I had more time, I could have written cleaner code.

There were a couple of features that I did not have time to implement, but should be easy to add in a subsequent version. The first is tracking cards that the user has revealed to particular players and displaying it in the notebook print out. This can help the user minimize the amount of information they give to the other players. Second is allowing the user to modify the cards used in the game during board set up. Clue has many versions worldwide, and the cards can vary slightly between them; for example, the app version of Clue allows you to pick your six suspects from up to 18 different characters.

I also did not end up having time to implement the bonus feature I mentioned in my design document. This bonus feature would incorporate some of the strategies I use when I play Clue to recommend which suspect, weapon, and room the user should use in their next suggestion.

Testing Tips

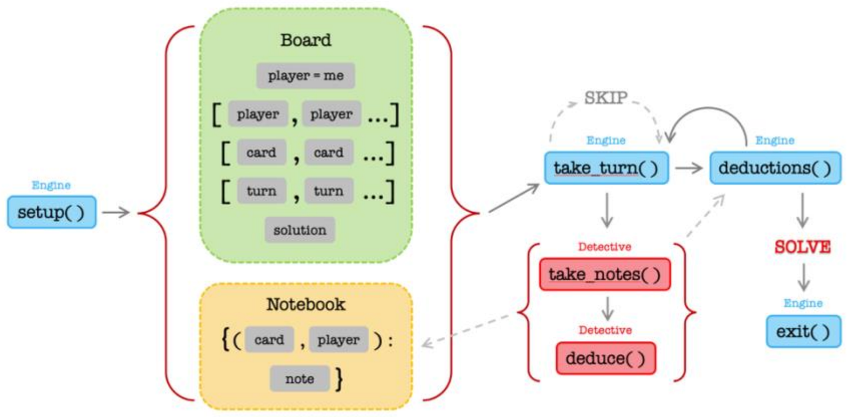
For me, the best (read: most fun ☺) way to test this program is to actually use it to try to win a game of Clue. The app version (https://www.marmaladegamestudio.com/games/clue/) is ideal, since you can play in single player mode against the computer and test games with different numbers of players without actually convincing that many friends to play Clue with you.

Short of this, however, I provided project1_testing_lea.pdf in the same folder for more detailed guidance.

Final Design Overview

Since I ended up changing my proposed implementation quite a bit, I'm including a brief overview of the final design to help with testing.

As mentioned in the beginning, I divided my code into five modules. The first three (`clue.py`, `board.py`, and `detective.py`) contain all the classes. The last two (`errors.py`, and `printouts.py`) only contain helper functions for the Engine class to help with its readability.



The program begins with an instance of the Engine class, which initializes with a set up function. The set up function creates an instance of the Board class, which stores the specifics of the Clue game being played. It also creates an instance of the Notebook class with all the cards in the player's hand marked.

Next, the engine guides the user through inputting a turn. The inputted turn is then passed on to an instance of the Detective class, which updates the notebook with information learned from the turn. The detective also uses this information to deduce any new information from previous notes.

After the detective is done, the engine looks at the notebook and determines if a solution is reached. If no solution or only a partial solution is reached, the engine prints out the results of the deduction along with the updated notes and takes another turn. If the full solution is reached, the engine it out and exits the program.