Project 3 - Web APIs

Due Date: Nov 11th, 2016
Any changes made to the assignment after posting it will be in red.

Overview

For this project, you are going to build a program that uses a web API to run a game of 1-2 Texas Holdem Poker. If you don't know how to play, here is an explanation. We will also discuss it in class.

One of the primary programming skills you will need in today's software developer landscape is working with web APIs. There are a lot of fun, free API's out there for you to use. Here is a quick tutorial on how accessing API's works in Ruby, and I will also cover the topic in class.

You will need to create Player, Card, Deck, and Table classes. The Deck class is just an aggregate of card and the Table class is an aggregate of Players, so you should implement the iterator pattern by including the Enumerable module.

You are also going to create a local flat database to store persistent statistical data on all the players.

Program Flow

The program will run as follows:

- First you will need to ask the user how many players should sit at the table with a minimum of 2 and maximum of 8.
- Ask the user what the starting chip amount is.
- Create players matching the number the user chose. You will need to generate names for each player and a play style and store the information in a Player class.
    - Play Style:
        i. Each computer player can choose from one of 4 play styles. Whether or not a player plays a hand is dependant on hand odds, pot size, and bankroll. You can determine the exact numbers yourself, but they should follow these guidelines.
            1. Loose Aggressive

        a. Plays low odds hands with a low bankroll or a low pot size.
    2. Loose Passive
        a. Plays high odds hands with a low bankroll or a low pot size.
    3. Tight Aggressive
        a. Plays low odds hands with a high bankroll or a high pot size.
    4. Tight Passive
        a. Plays high odds hands with a high bankroll or a high pot size.
  - **Extra Credit:** Use the Random User Generator API to get some of your user information. You must build a separate proxy class for this API if you use it.
- Once you start, you will need to randomly select the starting dealer, create a deck of cards, then begin playing.
  - The 'dealer' should move to the left by one person every hand.

The API

Note: The API should be fully working. If you run into any errors, please send me the url request you sent to the api.

For each players turn, you will make a call to the following api:
http://stevenamoore.me/projects/holdemapi?cards=<hand>&board=<board>&num_players=<number of players>

- If you are try to get player odds
  - cards: the two cards in the player's hand as a string in the following format: [AKQJT98765432][scdh][AKQJT98765432][scdh]
  - board <optional>: the shared cards on the board, if any. They follow the same format as the players cards. Must be between 3-5 cards.
  - num_players <optional - default is 2>: the number of players still in the hand at the table.
    i. *example call*

stevenamoore.me/projects/holdemapi?cards=AsKs&board=Ts9sJs&num_players=4
response: {"odds": 0.9162270439078798, "board": "Ts9sJs", "hand": "AsKs"}

- If you are trying to determine the winner

    - cards: the two cards in each of the players' hands as a string in the following format: [AKQJT98765432][scdh][AKQJT98765432][scdh]...
    - board: the shared cards on the board, if any. They follow the same format as the players cards. Must be 5 cards.
- The APi call will return a JSON response with the winning hand or hands in case of a tie
- example call

stevenamoore.me/projects/holdemapi?cards=AsKs2d5c&board=Ts9sJs
response: {"odds": 1, "board": "Ts9sJs", "hand": "AsKsTs9sJs"}

The more you use web API's, the more you will notice that all API calls work similarly. You should create an Proxy API superclass or module (your choice) that can be included in your Poker classes. You should then inherit or include your API call in your code.

You must also create a fallback in your Proxy class in case the api call fails. For example, if the poker api call fails, you could return a random percentage and a console message that the api call failed; however, the game should continue. There must also be some notification that the call failed.

Database

Once you have the game running, you should gather user statistics as you play and store it in a JSON file flat database.

You need to keep track of how many hands each player:

- folds,
- wins
- loses
- number of games played

You will need a database class that encapsulates writing to the database flat file.
If you are unfamiliar with flat file databases, a flat file database is a database that stores data in a plain text file. It's no different than storing it in a standard text file, except the data is structured. Each line of the text file holds one record, with fields separated by delimiters, such as commas or tabs. For our purposes (and to make things easier), we are going to store the data in the JSON format.

Completing the Game

After the game is done, you should print the winner, and the information for each player stored in the database. This should contain information across multiple games.

Changing the Assignment

You can change the assignment in any way you wish to make the assignment more interesting for extra credit. Any changes made to the requirements to make the program more sophisticated or interesting must be accompanied by an explanation in the Readme. The only rule is that the changes must not simplify the project.

Submission

- Required code organization:
    - Project3
        - Readme
            - Create a readme file containing an explanation of your project structure and any additional information you may need to add, and add it to your project folder
        - Required Classes
            - APIProxy.rb
                - An Abstract Class that should work for any API call with a JSON response
            - PokerOddsProxy.rb
                - Concrete class that derives from APIProxy and makes a call the the supplied holdemdapi
            - Database.rb
                - Encapsulates the Database functionality, such reading in the JSON file, writing the JSON file. Updating the database objects.
            - Player.rb
                - Contains player's name, cards, and bankroll
                - You may use this class for the user player or create another class if you wish

        - Card.rb
            - Contains the card's suit and value
        - Deck.rb
            - An aggregate class that contains 52 cards at the start of each game.
            - The cards should be shuffled before each game.
            - Should use the Enumerable Module to iterate through the deck
        - Table.rb
            - An aggregate class that contains up to 8 players.
            - Should also keep track of who is dealing, the current pot, who is in the game, who has folded, etc.
            - Should use the Enumerable Module to iterate through the Players
        - main.rb
        - **Extra Credit:** UserDataProxy.rb
        - …
            - Any additional classes must go into their own file
    - ***All file inclusions must be case sensitive. If you are working on a Windows machine, you will need to double check this.***
- Create a zip archive of your project 3 folder with the following command
    - zip -r project3 <<foldername>>
        - This creates an archive of all file and folders in the current directory called project3.zip
- Upload the archive to Blackboard under Project 3.
- You may demo your lab by downloading your archive from Blackboard. Extract your archive, then run your code, show your source, and answer any questions the TA may have.

Grading Guidelines

Total: 55 Points

- **Project:**
    - Allows User Player setup selections as described (2 points)
    - Assigns player types to each player (3 points)
    - Each computer player checks, bets, or folds based on player type and API call results (8 points)
    - User can select to check, bet, or fold, on each round and the appropriate action is taken (10 Points)

    - Dealer rotates around the table on each hand (3 points)
    - Uses Enumerable to iterate through the Players and Cards (2 points)
    - Uses the proxy pattern with an abstract superclass to access each API (8 points)
    - Has fallback for APIs in case of call failure (2 points)
    - Uses the 'open_uri' or Net::HTTP module from the Ruby library (2 points)
    - Uses the 'JSON' module from the Ruby library (2 points)
    - Database stores user statistics as a flat file (6 points)
    - All classes in a separate file (2 points)
    - **Extra Credit**
        - Uses Random User API(5 points)
- **Submission (5 points):**
    - Follows requested project structure and submission format (2 points)
    - Follows formatting guidelines (3 points)