DAT171 - Computer assignment 3 (2021-02-15)

The objective is to make use of the library from the previous assignment in a graphical user interface for a Texas Hold 'em game. The design must follow the Model-View-Controller concept. It should be possible to play the game until losing or winning. There need to be at least 2 players.

The first step is to create a suitable layout for buttons, players, displaying card data, etc.

1. Make a sketch (on paper) of how you want to display your Poker game (do a simple "boxy" design). Add sections for all the data you wish to display, for example, each player's hand and money, the cards on the table and the pot, bet, and fold buttons, and whatever you think you may need. Think through how you want this game to work on paper first instead of fixing things later. You must make a reuseable view for each player, so you should allocate a rectangular area that displays all the player information. *Tip: It is easier to only have one setup of input buttons (bet, fold etc.), representing the action for the active player.*
2. Find suitable QT-layouts to construct your sketch. Draw these on paper as well, to get a good idea of the hierarchies of the layouts and in what order to construct the widgets. You **must** make a (reusable) player view widget that displays a player model.
3. Write the code which draws up your layouts (in a single window) and add all the necessary buttons etc. Use subclasses and overload for each region of the screen (e.g. input control, player information, card areas). The buttons don't need to do anything yet.

See Lecture 11 for some examples of layouts, and links to even more tutorials and examples on the available layouts. Primarily, you will probably want to use the HBox and VBox layouts, but feel free to investigate other layouts (like the grid layout).

The second step is to model the game state, and connect it to the GUI code:

1. Make a class (or classes) that describes the game state. You don't need to support more than 2 players, but some simple betting with win/lose conditions is required. Think of a game state as the information needed to save the game (who the active player is, which cards are on the table, the pot, etc.). Make methods for betting/folding etc. that manipulate this game state. E.g. a class `TexasHoldEm` with the methods `fold`, `call`, `bet` could be suitable. You must also create a class representing a player. Avoid using enumerated variables such as `hand1, hand2, player1, player2`, use lists or tuples instead.
2. Connect the GUI widgets to the models, and add update functions in the view which display the current state of the model (player cards, money, pot, player turn, etc.). You must use signals.
3. Make the game playable, with a definite win/lose state (a popup message is sufficient for notifying losses/victories). It is recommended to make a signal for alerting the GUI that the game has ended. You should be able to play several rounds of poker until a player runs out of money.

See the `card_view.py` (available on the homepage) example for the split between model and view. You are free to use any of the code from the card view widget, modify it to fit your game (or not use it at all).

You can do basic stuff like checking user input for correctness in the GUI-code, but keep a clear separation of the graphical interface and the underlying model (game state).

Deliverables

Please submit a zip with

1. Your `cardlib` from the previous assignment.
2. `pokermodel.py` that contains all the game logic. This class may not import anything from your `pokerview` and only `QObject` and `pyqtSignal` from PyQt.
3. `pokerview.py` that contains all the GUI elements. This class may import `pokermodel` if needed.
4. `pokergame.py` the (short) main script that launches a game.

If you use the graphical components, please also include the directory with graphics `cards` in your zip. I should be able to run your code.

Signals must be used for communication. It must follow an MVC-design, so do not embed game logic in the GUI-code, and don't call GUI-code from the game logic!