Introduction and Explanation of Model, View, Controller Approach

Dear technically capable reader,

Thank you for taking interest in the internal workings of our Poker Game. The following will be a detailed overview of how these workings are structured, making use of some fairly advanced programming terms and constructs in order to best explain how we created our game.

First, it will be helpful to understand a few different programming techniques/approaches that our team utilized in order to meet our ultimate goal: a polished Texas Hold'em Poker computer game. One of these techniques was the Model View Controller methodology, which is an effective and intuitive way of structuring our code. This technique consisted of creating three separate groups of code, each containing several classes. These groups of code are stored as separate "packages" (a general grouping of classes for some related purpose). These three packages are named (not surprisingly) Model, View, and Controller.

The Model package contains classes which keep track of game information, providing the classes and methods necessary to set up and progress the game. The classes of the Model package operate purely on the raw data of the program. This means that these methods do not modify the GUI.

The GUI is setup through the View package. This package creates a visual display for the user, and provides the necessary methods to update the display..

Lastly, the Controller package links these other two components. The Controller package updates the display using information from the Model, meanwhile using the methods of the Model to progress the game.

A useful tool that we used to create our GUI was the JavaFx scene builder, which provided a relatively simple way of putting together our beautiful GUI. JavaFx is quite similar to Java Swing, but contains more features.

Step-by-Step Execution with Analysis of Object Oriented Approach

Now, we will guide you through the step-by-step execution of our program, explaining how each object in our Object Oriented Design fits together to provide all necessary behaviors and changes of state needed to progress through a game of Texas Hold'em Poker.

As is probably clear from the Model, View, Controller segment of the Introduction, the centerpoint for the execution of our game is the Controller package, and namely, the MainController class. As explained in the introduction, the MainController uses information from the Model to update the View (the GUI).

MainController, however, is not the only class in the Controller package. There are also AIController classes. We structured our controller classes the AI can make decisions based on the state of the game. Secondly, there is the MainPageController, which is only initialized for the display of the start menu. This accesses user input from the start menu and passes these options into the Model. Now that the structure of our controller package has been explained, we can now begin explaining the execution of our game, starting with the role of the MainController, and branching off as necessary.

The first thing that our MainController object must do is to initialize the AI, Player, and GameModel objects. The GameModel is the central class of the Model package. Creating a GameModel object automatically sets up a Deck containing 52 Cards.This is the software representation of a standard deck of cards. The GameModel then deals two cards to each player's Hand. The GameModel provides methods for choosing one of the standard Poker moves (fold, call, raise, check) for each round, and for each player, as the common cards are revealed.

After these components have been set up, our MainController class must create the AIController objects mentioned briefly in the paragraph before the last. These controllers read information from the GameModel class in order to update the state (decision) of an AI. This means that we will have three separate AIController objects, one for each AI object. The AIController only reads information from the Model. It never changes the state of the GameModel object, and never updates the View. It's sole purpose is to choose moves for the AI. It does this by first gathering the cards from the AI's hand and the pool of common cards. Once this is done, the AI then determines all possible cards which could become part of the pool of common cards in the subsequent rounds. For each situation, the AI scores its best hand, and adds this score to a running total. The AI then chooses its move based on this total.

Now that the MainController has set up the GameModel, AI's, Player, and AIController's, it is now ready to start iterating through the game, using the GameModel's methods to advance the game and receive input from the AI's. The human Player's decisions

are processed through input into the MainController from the GUI. The game progresses turn by turn, until a winner is determined and displayed.

Analysis of User Stories - How was each story implemented?

- As a player, I want to play TEXAS HOID ‘EM
    - Explained throughout document.
- As a player, I want to be dealt a set of cards so that I can decide on my moves (randomly from deck of 52)
    - This was done by creating a Deck class, which initializes 52 Cards, and uses Java’s Random class to choose (and remove) cards from the deck and add them to the player’s Hand.
    - The GUI then displays the cards (general/special card view) in the hand, and the player can choose their moves with a set of buttons.
- As a player, I want to have options to bet, fold, call and raise.
    - This was accomplished with a set of buttons in the GUI which, when pressed, pass relevant information to the Player class to that it can update its state, and the GameModel can use the Player’s state to progress the game.
- As a player, I want to have a visually stunning GUI. I can enjoy my game
    - This was done by utilizing the extensive JavaFX library to build our GUI.
    - We create 2 screens, main page and game view, which have several visual effects. The buttons is also included to let users go from one to another.

- As a player, I want to be able to drag and drop card in GUI. So that I can rearrange the card.
    - This was done by generating several mouse-event handlers (or controllers) which are able to detect the gesture of drag-and-drop from user, and then implement it (switch two cards).
- As a player, I want the game can let me play turn by turn.
    - The GameModel class keeps track of whose turn it is, and once a player chooses their move, the GameModel moves on to the next player.
- As a player, I want to know how much money do I have.
    - This was done by showing a colored label on the top right corner of the GUI.
- As a player, I want to play with other people(via network) or AI(Artificial Intelligence)
    - We implemented AI but not network play. The operation of the AI in our game is explained extensively in the previous section.
- As a player, I want a range of difficulty options.
    - This was accomplished by overriding the constructor of the AIController, allowing the AIController to take on different “personalities” based on a parameter (an enum) which gives three options. We didn’t have the time to fully implement this feature into the game, however, so a default AI personality is utilized in our version 1.0 of the game.
- As a player, I want to be able to choose custom sets of cards (maybe even unlock sets, as a perk)
    - We implemented this feature through special Bucknell Faculty face cards, which replace the default face cards.
- As a player, I want a smooth, stable, and intuitive gaming experience.

    - This was accomplished by...TESTING, TESTING, TESTING!
- As a player/programmer, I want clear UML documentation.
- As a programmer, I want test classes to diagnose bugs with.
- As a player, I want to see the winner at the end of the game.
    - This was implemented by showing the name of the winner in the middle of screen when the game is over.
- As a player, I want to hear applause if I win.
    - While we did not get a chance to implement a sound for victory, we were able to add a set of sounds to the GUI buttons. This was accomplished by passing a WAV file to a SoundPlayer class, which contained a method called playSound that plays the WAV.