- Virtual Poker Chips
    - Junyang Lu
    - Nguyen Nguyen
    - Ayushman Satpathy
    - Gabriella Wang
    - Christian Michael Dela Cruz

We can analyze the function points for each method defined in the use case for placing a bet by first listing the components for the software as a whole.

- External Input (EI)
    - Data coming from the user, such as submitting the bet or folding
- External Output (EO)
    - Data heading out, such as success or fail message when placing bets
- External Inquiry (EQ)
    - Request and response with no internal data modification, which includes all the intermediate validation steps
- Internal Logical File (ILF)
    - The data that gets maintained internally, such as the player profiles, current chip balance, the ongoing game sessions, etc...
- External Interface File (EIF)
  - The data that gets used by the software from a third party. This is not very applicable to our use case
  Then, we can apply the weighting scale mentioned in class to each of the methods for this specific use case.

createValidationRequest()

Summary: sends an API request to the backend as the frontend button "place bet" is clicked

- EI
    - User input to validate the bet they just made
    - This is simple to implement
    - 3
- EO
    - Returns the results of the bet validation
    - This is also of low complexity
    - 4
- EQ
    - This is the input and output requests with no database updates
    - Simple to implement as well

    - 3
- ILF
    - None
- EIF
  - None
  Total: 3 + 4 + 3 =10

validateBet()

Summary: checks whether the bet amount specified by the player is valid. For example, it cannot be negative or 0, and does not exceed the current number of chips the player has

- EI
    - Inputs represent player's desired amount of chips to bet
    - Simple complexity
    - 3
- EQ
    - Validates the bet by performing type check and basic numerical comparisons, nothing is getting written into any database
    - Simple
    - 3
- EO
    - Only true or false outputs
- ILF
    - Uses cached player chip counts, stored on the host as an instance of the game session, very simple storage mechanism and very simple read
    - Not a lot of overhead
    - 7
- EIF
  - None
  Total: 3 + 3 + 7 = 13

recordPlayerBet()

Summary: records the player bet in the database

- EI
    - Only takes player ID, bet amount, and game session ID as input
    - Easy to implement
    - 3
- EQ
    - None
- ILF

    - Writes to the player bets to an internal database with aforementioned information
    - Needs to connect to database and ensure insertion success, average difficulty
    - 10
- EO
    - Confirms write success to the database
    - Relatively easy to implement
    - 4
- EIL
  - None
  Total: 3 + 10 + 4 = 17

broadCastBetToServer()

Summary: sends the bet details of a player to the server database to be recorded and also propagates this change to other players in the same game session (storing bet information game-wide)

- EI
    - Sends the bet information from each player to the server/host via WebSocket
    - Information is easy to send
    - 3
- ILF
    - Updates the state for all players, as each player makes the bet, and stores all relevant information
    - Information synchronization and WebSocket initialization/utilization is more complicated
    - 10
- EO
  - Broadcasts the update to all connected players
  - WebSocket and synchronization complexity
  - 10
  Total: 3 + 10 + 5 = 18