**Classes**:

1. **Card:**

Fields:

1. Value
2. Suit

Methods:

1. \_\_str\_\_ () - prints the card in the way instructed.
2. **Deck**:

Fields:

1. Deck – each object created generate a 52-card deck using 2 dictionary object I have defined according to the instructions.

Methods:

1. deal – returns a card from the front of the deck and deletes it
2. \_\_len\_\_ - return the length of the deck (number of cards) – can use len function on deck object
3. deal\_hands – split the deck object into 2 even list’s and return’s them as a 2-tuple
4. shuffle – shuffles the deck using random.
5. **War (list a, list b)**

Supports one game session. Designed for 2 equal lists of cards to start a game session.

To create an object, pass 2 equal length lists of cards.

Methods:

1. game\_check() – check’s if there are any cards left on the deck.
2. compare\_cards() – compare between the player and the server card’s fields and return a value according to the outcome.
3. Deal() – deal 1 round of the game and returning the player’s card.
4. Update(string) – update’s the player\_request field according to data given.
5. create\_msg() – return’s the correct message according to the player\_request field Each message type has its own header
6. war – handles the war situation according to the instructions.
7. getCard() – return’s the card in the front of the player’s deck.
8. bad\_input() – return’s a string that a bad input has been given by the player
9. **Server (address, port)**

To create a server object, pass the host (IP address) and port number.

Fields:

a. host – the Ipv4 of the device.

b. port – the port number to bind the created socket to,

c. s – socket object initialized as None.

Methods:

a. create\_socket() – create and assign socket to the s field.

b. bind\_socket() – bind the socket to the port selected in the main function and listen for connections. return true if successful else return False

d. accepting\_connection() – loop on an infinite while loop to accept client connection. If there are less, then 2 players playing – accepts the connection and start’s a game session else: send denied message to the client. Even if game’s is in session- keep looking for connections. Idle client is defined as 1 who didn’t send any message to client within 30 seconds.

3. threaded\_client(clients addr) – handles each client’s game session. the game logic and messaging is with in this function.

1. **Client (address, port)**

Fields:

a. host – server’s Ip

b. port – server’s port

c. s – client socket

Methods:

a. read\_input(string) – read and handles input from the player. Adjusting it to the war game class.

b. print\_info() – return’s a string that explains the correct input’s the player should give while playing.

c. making\_connection() – create a socket for the client and making a connection with the server. If accept msg given – activates handling\_connection method.

d. handling\_connection() – hold the logic to handle server data given while playing.