Project Description:

I'm developing a blackjack game where I connect to a server to play against the computer dealer. In this single-player game, I'll be able to place bets, receive cards, and make decisions against the dealer.

Project Goal:

My goal is to showcase my understanding of socket programming in Python while implementing the game logic for a single-player blackjack game.

Code Structure:

To achieve this, I'll be structuring my code as follows:

- Loops: I'll use loops for repetitive tasks such as dealing cards, managing player turns, and controlling the flow of the game.
- Functions: I'll organize my code into functions to handle different aspects of the game, such as dealing cards, calculating hand values, and determining winners.
- Lists and Dictionaries: Utilizing lists and dictionaries, I'll represent decks of cards, player hands, and the game state.
- Files: I'll use files to implement the functionality to save game data to files.
- Strings: Manipulating strings will allow me to display messages to the player, including game instructions, player actions, and game results.
- Socket Programming: Implementing socket programming will establish communication between the game client and server, enabling me to interact with the game.

Project Source code:

```python
# server.py
from socket import socket
from random import shuffle
import os

HOST = "localhost"
PORT = 5000
ADDRESS = (HOST, PORT)
SAVE_DIRECTORY = "player_data"

class Blackjack:
    def __init__(self):
        self.deck = []
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        values = list(range(2, 11)) + [10, 10, 10, 11]
        for suit in suits:
            for value in values:
                self.deck.append((value, suit))
        shuffle(self.deck)
        self.player_hand = [self.deck.pop()]
        self.dealer_hand = [self.deck.pop(), self.deck.pop()]
        if self.player_hand[0][0] == 11 and self.dealer_hand[0][0] == 11:
            self.player_hand.append((1, self.player_hand[0][1]))

    def get_game_state(self):
        return f"Player's hand: {self.player_hand}, Total: {self.calculate_hand_total(self.player_hand)}\nDealer's visible card: {self.dealer_hand[0]}"

    def hit(self, hand):
        hand.append(self.deck.pop())

    def stand(self):
        while self.calculate_hand_total(self.dealer_hand) < 17:
            self.hit(self.dealer_hand)

    def is_game_over(self):
        player_total = self.calculate_hand_total(self.player_hand)
        if player_total == 21:
            return True, "You win! You have blackjack."
        elif player_total > 21:
            return True, "You bust! Dealer wins."
        dealer_total = self.calculate_hand_total(self.dealer_hand)
        if dealer_total == 21:
            return True, "Dealer wins! Dealer has blackjack."
        elif dealer_total > 21:
            return True, "Dealer busts! You win."
        return False, None

    def calculate_hand_total(self, hand):
```

```
        total = sum(card[0] for card in hand)
        num_aces = sum(1 for card in hand if card[0] == 11)
        while total > 21 and num_aces > 0:
            total -= 10
            num_aces -= 1
        return total

def save_game_data(player_name, game_data):
    if not os.path.exists(SAVE_DIRECTORY):
        os.makedirs(SAVE_DIRECTORY)
    with open(os.path.join(SAVE_DIRECTORY, f"{player_name}.txt"), "w") as file:
        file.write(game_data)

s = socket()
print("Socket created successfully!")
s.bind(ADDRESS)
s.listen()
print("Server is listening for connections...")

while True:
    print("Waiting for connection...")
    (client, address) = s.accept()
    print(f"Connection established with {address}")

    # Ask for player's name
    client.send("Enter your name: ".encode())
    player_name = client.recv(1024).decode()
    print(f"Player's name: {player_name}")

    # Create a Blackjack instance for this client
    blackjack_game = Blackjack()

    while True:
        client.send("Welcome to the blackjack game! Let's start.\n".encode())

        while True:
            client.send((blackjack_game.get_game_state() + "\n").encode())
            action = client.recv(1024).decode().lower()
            if action == "hit":
                blackjack_game.hit(blackjack_game.player_hand)
            elif action == "stand":
                blackjack_game.stand()
                blackjack_game.hit(blackjack_game.dealer_hand)
                break

            game_over, message = blackjack_game.is_game_over()
            if game_over:
                client.send("Game over\n".encode())
                client.send((message + "\n").encode())
                save_game_data(player_name, message)
                break

        game_over, message = blackjack_game.is_game_over()
        if game_over:
            client.send("Game over\n".encode())
            client.send((message + "\n").encode())
            play_again = client.recv(1024).decode().lower()
            if play_again != "yes":
                break
            else:
                blackjack_game = Blackjack()

    client.close()

s.close()
```