

You

Create a python game based on the classic blackjack casino game, with a proper card set.

GPT-4o

Below is a simple implementation of a Blackjack game in Python. This game uses a standard deck of cards and allows a player to play against the dealer (computer). The game follows the basic rules of Blackjack, where the objective is to get a hand value as close to 21 as possible without exceeding it.

```
import random

# Define card values
card_values = {
    '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
    'J': 10, 'Q': 10, 'K': 10, 'A': 11
}

# Create a standard deck of cards
deck = [rank for rank in card_values.keys()] * 4

def deal_card(deck):
    """Deals a card from the deck."""
    return deck.pop(random.randint(0, len(deck) - 1))

def calculate_hand_value(hand):
    """Calculates the total value of a hand."""
    value = sum(card_values[card] for card in hand)
    # Adjust for Aces if value is over 21
    ace_count = hand.count('A')
    while value > 21 and ace_count:
        value -= 10
        ace_count -= 1
    return value

def display_hand(player, hand):
    """Displays the player's or dealer's hand."""
    print(f"{player}'s hand: {' '.join(hand)} (Value: {calculate_hand_value(hand)})")

def blackjack():
    # Shuffle the deck
    random.shuffle(deck)
```

```

# Initial deal
player_hand = [deal_card(deck), deal_card(deck)]
dealer_hand = [deal_card(deck), deal_card(deck)]

# Display initial hands
display_hand("Player", player_hand)
print(f"Dealer's hand: {dealer_hand[0]}, ?")

# Player's turn
while True:
    choice = input("Do you want to 'hit' or 'stand'? ").strip().lower()
    if choice == 'hit':
        player_hand.append(deal_card(deck))
        display_hand("Player", player_hand)
        if calculate_hand_value(player_hand) > 21:
            print("Player busts! Dealer wins.")
            return
    elif choice == 'stand':
        break
    else:
        print("Invalid choice. Please choose 'hit' or 'stand'.")

# Dealer's turn
display_hand("Dealer", dealer_hand)
while calculate_hand_value(dealer_hand) < 17:
    dealer_hand.append(deal_card(deck))
    display_hand("Dealer", dealer_hand)
    if calculate_hand_value(dealer_hand) > 21:
        print("Dealer busts! Player wins.")
        return

# Determine winner
player_value = calculate_hand_value(player_hand)
dealer_value = calculate_hand_value(dealer_hand)
if player_value > dealer_value:
    print("Player wins!")
elif player_value < dealer_value:
    print("Dealer wins!")
else:
    print("It's a tie!")

if __name__ == "__main__":
    blackjack()

```

How to Play

1. Run the script in a Python environment.
2. The player is dealt two cards, and one of the dealer's cards is shown.
3. The player can choose to "hit" (take another card) or "stand" (end their turn).

4. The dealer will then reveal their hand and draw cards until their hand value is at least 17.
5. The winner is determined by who has the highest hand value that does not exceed 21.

Game Rules

- Number cards (2-10) are worth their face value.
- Face cards (J, Q, K) are worth 10 points.
- Aces can be worth 1 or 11 points, whichever is more favorable for the hand.
- The player wins if their hand value is closer to 21 than the dealer's hand value, without exceeding 21.
- If a hand value exceeds 21, it is a "bust," and the player or dealer loses.

You

Using your working game file adjust the following: Make the playing cards objects and assign them the values needed, also change the hit or stand question to expect 'h' and 's' for hit and stand.

GPT-4o

To adjust the game using object-oriented programming, we can create a **Card** class to represent each card in the deck. Additionally, we'll modify the input for hitting or standing to expect 'h' for hit and 's' for stand. Below is the updated implementation:

```
import random

class Card:
    """Represents a single playing card."""
    def __init__(self, rank, value):
        self.rank = rank
        self.value = value

    def __str__(self):
        return self.rank

class Deck:
    """Represents a deck of playing cards."""
    def __init__(self):
        self.cards = []
        self.build_deck()

    def build_deck(self):
        """Builds a standard deck of 52 cards."""
        card_values = {
            '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
            'J': 10, 'Q': 10, 'K': 10, 'A': 11
        }
```

```

        'J': 10, 'Q': 10, 'K': 10, 'A': 11
    }
    for rank, value in card_values.items():
        for _ in range(4): # Four of each rank
            self.cards.append(Card(rank, value))

    def shuffle(self):
        """Shuffles the deck."""
        random.shuffle(self.cards)

    def deal_card(self):
        """Deals a card from the deck."""
        return self.cards.pop()

    def calculate_hand_value(hand):
        """Calculates the total value of a hand."""
        value = sum(card.value for card in hand)
        ace_count = sum(1 for card in hand if card.rank == 'A')
        while value > 21 and ace_count:
            value -= 10
            ace_count -= 1
        return value

    def display_hand(player, hand):
        """Displays the player's or dealer's hand."""
        hand_str = ', '.join(str(card) for card in hand)
        print(f"{player}'s hand: {hand_str} (Value: {calculate_hand_value(hand)})")

    def blackjack():
        # Initialize and shuffle the deck
        deck = Deck()
        deck.shuffle()

        # Initial deal
        player_hand = [deck.deal_card(), deck.deal_card()]
        dealer_hand = [deck.deal_card(), deck.deal_card()]

        # Display initial hands
        display_hand("Player", player_hand)
        print(f"Dealer's hand: {dealer_hand[0]}, ?")

        # Player's turn
        while True:
            choice = input("Do you want to 'hit' (h) or 'stand' (s)? ").strip().lower()
            if choice == 'h':
                player_hand.append(deck.deal_card())
                display_hand("Player", player_hand)
                if calculate_hand_value(player_hand) > 21:
                    print("Player busts! Dealer wins.")
                    return
            elif choice == 's':

```

```

        break
    else:
        print("Invalid choice. Please choose 'h' or 's'.")

# Dealer's turn
display_hand("Dealer", dealer_hand)
while calculate_hand_value(dealer_hand) < 17:
    dealer_hand.append(deck.deal_card())
    display_hand("Dealer", dealer_hand)
    if calculate_hand_value(dealer_hand) > 21:
        print("Dealer busts! Player wins.")
        return

# Determine winner
player_value = calculate_hand_value(player_hand)
dealer_value = calculate_hand_value(dealer_hand)
if player_value > dealer_value:
    print("Player wins!")
elif player_value < dealer_value:
    print("Dealer wins!")
else:
    print("It's a tie!")

if __name__ == "__main__":
    blackjack()

```

Changes Made

1. **Card Class:** Created a **Card** class to represent each card, with attributes for rank and value.
2. **Deck Class:** Created a **Deck** class to manage the deck of cards, including building and shuffling the deck.
3. **Input Handling:** Changed the input for hitting or standing to expect 'h' for hit and 's' for stand, making it more intuitive and quicker for the player.
4. **Game Logic:** Adjusted the game logic to work with card objects instead of string representations.

You

Now use the given code and add an welcoming message at the beginning of the game as well as an play again option after ending the complete round.

GPT-4o

To enhance the game with a welcoming message and a "play again" option, we'll add a few more features to the existing code. Here's the updated implementation:

```

import random

class Card:
    """Represents a single playing card."""
    def __init__(self, rank, value):
        self.rank = rank
        self.value = value

    def __str__(self):
        return self.rank

class Deck:
    """Represents a deck of playing cards."""
    def __init__(self):
        self.cards = []
        self.build_deck()

    def build_deck(self):
        """Builds a standard deck of 52 cards."""
        card_values = {
            '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
            'J': 10, 'Q': 10, 'K': 10, 'A': 11
        }
        for rank, value in card_values.items():
            for _ in range(4): # Four of each rank
                self.cards.append(Card(rank, value))

    def shuffle(self):
        """Shuffles the deck."""
        random.shuffle(self.cards)

    def deal_card(self):
        """Deals a card from the deck."""
        return self.cards.pop()

    def calculate_hand_value(hand):
        """Calculates the total value of a hand."""
        value = sum(card.value for card in hand)
        ace_count = sum(1 for card in hand if card.rank == 'A')
        while value > 21 and ace_count:
            value -= 10
            ace_count -= 1
        return value

    def display_hand(player, hand):
        """Displays the player's or dealer's hand."""
        hand_str = ', '.join(str(card) for card in hand)
        print(f"{player}'s hand: {hand_str} (Value: {calculate_hand_value(hand)})")

    def blackjack():

```

```

print("Welcome to the Blackjack Game!")

while True:
    # Initialize and shuffle the deck
    deck = Deck()
    deck.shuffle()

    # Initial deal
    player_hand = [deck.deal_card(), deck.deal_card()]
    dealer_hand = [deck.deal_card(), deck.deal_card()]

    # Display initial hands
    display_hand("Player", player_hand)
    print(f"Dealer's hand: {dealer_hand[0]}, ?")

    # Player's turn
    while True:
        choice = input("Do you want to 'hit' (h) or 'stand' (s)? ").strip().lower()

        if choice == 'h':
            player_hand.append(deck.deal_card())
            display_hand("Player", player_hand)
            if calculate_hand_value(player_hand) > 21:
                print("Player busts! Dealer wins.")
                break
        elif choice == 's':
            break
        else:
            print("Invalid choice. Please choose 'h' or 's'.")

    # If player hasn't busted, it's dealer's turn
    if calculate_hand_value(player_hand) <= 21:
        display_hand("Dealer", dealer_hand)
        while calculate_hand_value(dealer_hand) < 17:
            dealer_hand.append(deck.deal_card())
            display_hand("Dealer", dealer_hand)
            if calculate_hand_value(dealer_hand) > 21:
                print("Dealer busts! Player wins.")
                break

    # Determine winner if no busts
    if calculate_hand_value(player_hand) <= 21 and calculate_hand_value(dealer_h
and) <= 21:
        player_value = calculate_hand_value(player_hand)
        dealer_value = calculate_hand_value(dealer_hand)
        if player_value > dealer_value:
            print("Player wins!")
        elif player_value < dealer_value:
            print("Dealer wins!")
        else:
            print("It's a tie!")

```

```

        # Ask if the player wants to play again
        play_again = input("Do you want to play again? (y/n): ").strip().lower()
        if play_again != 'y':
            print("Thank you for playing! Goodbye!")
            break

if __name__ == "__main__":
    blackjack()

```

Enhancements

1. **Welcoming Message:** A welcoming message is displayed at the start of the game to greet the player.
2. **Play Again Option:** After each round, the player is asked if they want to play again. If the player inputs 'y', the game restarts; otherwise, it exits with a goodbye message.

You

Now add the functionality to play for in game money. every player should begin with 100\$ and cannot deposit any more money if he has none left. If the player loses, he will lose his stake, if he wins he will get the doubled amount back, if the game is a tie, the gets back his stake. Also tell the player his current amount of money left and ask for the amount to gamble with in this round.

GPT-4o

To add the functionality for playing with in-game money, we need to track the player's balance, allow them to place stakes, and adjust their balance based on the outcome of each round. Here's the updated implementation:

```

import random

class Card:
    """Represents a single playing card."""
    def __init__(self, rank, value):
        self.rank = rank
        self.value = value

    def __str__(self):
        return self.rank

class Deck:
    """Represents a deck of playing cards."""
    def __init__(self):
        self.cards = []
        self.build_deck()

```



```

def build_deck(self):
    """Builds a standard deck of 52 cards."""
    card_values = {
        '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 1
0,
        'J': 10, 'Q': 10, 'K': 10, 'A': 11
    }
    for rank, value in card_values.items():
        for _ in range(4): # Four of each rank
            self.cards.append(Card(rank, value))

def shuffle(self):
    """Shuffles the deck."""
    random.shuffle(self.cards)

def deal_card(self):
    """Deals a card from the deck."""
    return self.cards.pop()

def calculate_hand_value(hand):
    """Calculates the total value of a hand."""
    value = sum(card.value for card in hand)
    ace_count = sum(1 for card in hand if card.rank == 'A')
    while value > 21 and ace_count:
        value -= 10
        ace_count -= 1
    return value

def display_hand(player, hand):
    """Displays the player's or dealer's hand."""
    hand_str = ', '.join(str(card) for card in hand)
    print(f"{player}'s hand: {hand_str} (Value: {calculate_hand_value(hand)})")

def blackjack():
    print("Welcome to the Blackjack Game!")

    balance = 100 # Player starts with $100

    while balance > 0:
        print(f"Your current balance: ${balance}")

        # Ask for the stake amount
        while True:
            try:
                stake = int(input("Enter your stake for this round: $"))
                if 0 < stake <= balance:
                    break
            else:
                print(f"Invalid stake amount. You can bet between $1 and ${balan
ce}.")

        except ValueError:

```

```

        print("Please enter a valid number.")

# Initialize and shuffle the deck
deck = Deck()
deck.shuffle()

# Initial deal
player_hand = [deck.deal_card(), deck.deal_card()]
dealer_hand = [deck.deal_card(), deck.deal_card()]

# Display initial hands
display_hand("Player", player_hand)
print(f"Dealer's hand: {dealer_hand[0]}, ?")

# Player's turn
while True:
    choice = input("Do you want to 'hit' (h) or 'stand' (s)? ").strip().lower()

    if choice == 'h':
        player_hand.append(deck.deal_card())
        display_hand("Player", player_hand)
        if calculate_hand_value(player_hand) > 21:
            print("Player busts! Dealer wins.")
            balance -= stake
            break
    elif choice == 's':
        break
    else:
        print("Invalid choice. Please choose 'h' or 's'.")

# If player hasn't busted, it's dealer's turn
if calculate_hand_value(player_hand) <= 21:
    display_hand("Dealer", dealer_hand)
    while calculate_hand_value(dealer_hand) < 17:
        dealer_hand.append(deck.deal_card())
        display_hand("Dealer", dealer_hand)
    if calculate_hand_value(dealer_hand) > 21:
        print("Dealer busts! Player wins.")
        balance += stake
        break

# Determine winner if no busts
if calculate_hand_value(player_hand) <= 21 and calculate_hand_value(dealer_hand) <= 21:
    player_value = calculate_hand_value(player_hand)
    dealer_value = calculate_hand_value(dealer_hand)
    if player_value > dealer_value:
        print("Player wins!")
        balance += stake
    elif player_value < dealer_value:
        print("Dealer wins!")

```

```

        balance -= stake
    else:
        print("It's a tie!")
        # Balance remains unchanged

# Check if the player can continue
if balance <= 0:
    print("You have no money left to play. Game over!")
    break

# Ask if the player wants to play again
play_again = input("Do you want to play again? (y/n): ").strip().lower()
if play_again != 'y':
    print("Thank you for playing! Goodbye!")
    break

if __name__ == "__main__":
    blackjack()

```

Enhancements

1. **In-Game Money:** The player starts with \$100. The balance is displayed at the start of each round.
2. **Stake Input:** The player is prompted to enter their stake for each round, which must be between \$1 and their current balance.
3. **Balance Adjustment:**
 - If the player wins, their balance increases by the stake amount.
 - If the player loses, their balance decreases by the stake amount.
 - If the game is a tie, the stake is returned to the player.
4. **Game Over Condition:** The game ends if the player runs out of money.
5. **Play Again Option:** After each round, the player can choose to play again or exit the game.

You

Using the previous code, adjust the game to be playable by multiple players, by including a player class, with the name, balance and all other needed objects for a working game.

GPT-4o

To make the game playable by multiple players, we'll introduce a **Player** class to manage each player's attributes such as name, balance, and hand. This will allow us to handle multiple players in the game loop. Here's the updated implementation:

```

import random

```

```

class Card:
    """Represents a single playing card."""
    def __init__(self, rank, value):
        self.rank = rank
        self.value = value

    def __str__(self):
        return self.rank

class Deck:
    """Represents a deck of playing cards."""
    def __init__(self):
        self.cards = []
        self.build_deck()

    def build_deck(self):
        """Builds a standard deck of 52 cards."""
        card_values = {
            '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
            'J': 10, 'Q': 10, 'K': 10, 'A': 11
        }
        for rank, value in card_values.items():
            for _ in range(4): # Four of each rank
                self.cards.append(Card(rank, value))

    def shuffle(self):
        """Shuffles the deck."""
        random.shuffle(self.cards)

    def deal_card(self):
        """Deals a card from the deck."""
        return self.cards.pop()

class Player:
    """Represents a player in the game."""
    def __init__(self, name):
        self.name = name
        self.balance = 100
        self.hand = []

    def display_hand(self):
        """Displays the player's hand."""
        hand_str = ', '.join(str(card) for card in self.hand)
        print(f"{self.name}'s hand: {hand_str} (Value: {calculate_hand_value(self.hand)})")

    def calculate_hand_value(hand):
        """Calculates the total value of a hand."""
        value = sum(card.value for card in hand)
        ace_count = sum(1 for card in hand if card.rank == 'A')

```

```

while value > 21 and ace_count:
    value -= 10
    ace_count -= 1
return value

def blackjack():
    print("Welcome to the Multiplayer Blackjack Game!")

    # Create players
    num_players = int(input("Enter the number of players: "))
    players = [Player(input(f"Enter the name for player {i + 1}: ")) for i in range
(num_players)]

    while any(player.balance > 0 for player in players):
        # Initialize and shuffle the deck
        deck = Deck()
        deck.shuffle()

        # Clear previous hands
        for player in players:
            player.hand = []

        # Initial deal
        dealer_hand = [deck.deal_card(), deck.deal_card()]
        for player in players:
            player.hand = [deck.deal_card(), deck.deal_card()]

        # Players' turns
        for player in players:
            if player.balance <= 0:
                print(f"{player.name} has no money left to play.")
                continue

            print(f"\n{player.name}'s turn (Balance: ${player.balance})")
            player.display_hand()
            print(f"Dealer's hand: {dealer_hand[0]}, ?")

            # Ask for the stake amount
            while True:
                try:
                    stake = int(input(f"{player.name}, enter your stake for this rou
nd: $"))

                    if 0 < stake <= player.balance:
                        break
                    else:
                        print(f"Invalid stake amount. You can bet between $1 and ${p
layer.balance}.")
                except ValueError:
                    print("Please enter a valid number.")

            # Player's decision

```

```

        while True:
            choice = input("Do you want to 'hit' (h) or 'stand' (s)? ").strip().lower()

            if choice == 'h':
                player.hand.append(deck.deal_card())
                player.display_hand()
                if calculate_hand_value(player.hand) > 21:
                    print(f"{player.name} busts! Dealer wins.")
                    player.balance -= stake
                    break
            elif choice == 's':
                break
            else:
                print("Invalid choice. Please choose 'h' or 's'.")

        # Dealer's turn
        print("\nDealer's turn")
        while calculate_hand_value(dealer_hand) < 17:
            dealer_hand.append(deck.deal_card())
        display_hand("Dealer", dealer_hand)

        # Determine winners
        dealer_value = calculate_hand_value(dealer_hand)
        for player in players:
            if player.balance <= 0:
                continue
            player_value = calculate_hand_value(player.hand)
            if player_value <= 21:
                if dealer_value > 21 or player_value > dealer_value:
                    print(f"{player.name} wins!")
                    player.balance += stake
                elif player_value < dealer_value:
                    print(f"{player.name} loses.")
                    player.balance -= stake
                else:
                    print(f"{player.name} ties with the dealer.")
                    # Balance remains unchanged

        # Check if any player can continue
        if not any(player.balance > 0 for player in players):
            print("All players have run out of money. Game over!")
            break

        # Ask if the players want to play again
        play_again = input("\nDo you want to play another round? (y/n): ").strip().lower()

        if play_again != 'y':
            print("Thank you for playing! Goodbye!")
            break

def display_hand(player, hand):

```

```

    """Displays the dealer's hand."""
    hand_str = ', '.join(str(card) for card in hand)
    print(f"{player}'s hand: {hand_str} (Value: {calculate_hand_value(hand)})")

if __name__ == "__main__":
    blackjack()

```

Key Changes

1. **Player Class:** Introduced a **Player** class to manage each player's name, balance, and hand.
2. **Multiple Players:** The game now supports multiple players, each with their own balance and hand.
3. **Stake Management:** Each player can place a stake for each round, and their balance is adjusted based on the outcome.
4. **Game Loop:** The game continues as long as at least one player has money left. Players without money cannot participate further.
5. **Play Again Option:** After each round, players can choose to play another round or exit the game.

You

The code you created includes an game order mistake, the system is asking for the players stake after showing him the dealers and the players hand, player ask all players to put in there stakes before starting the round

GPT-4o

To correct the order of operations in the game, we need to ask each player for their stake before dealing the cards and showing the hands. This ensures that the players make their betting decisions without seeing any cards. Here's the updated code:

```

import random

class Card:
    """Represents a single playing card."""
    def __init__(self, rank, value):
        self.rank = rank
        self.value = value

    def __str__(self):
        return self.rank

class Deck:
    """Represents a deck of playing cards."""
    def __init__(self):

```

```

        self.cards = []
        self.build_deck()

    def build_deck(self):
        """Builds a standard deck of 52 cards."""
        card_values = {
            '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
            'J': 10, 'Q': 10, 'K': 10, 'A': 11
        }
        for rank, value in card_values.items():
            for _ in range(4): # Four of each rank
                self.cards.append(Card(rank, value))

    def shuffle(self):
        """Shuffles the deck."""
        random.shuffle(self.cards)

    def deal_card(self):
        """Deals a card from the deck."""
        return self.cards.pop()

class Player:
    """Represents a player in the game."""
    def __init__(self, name):
        self.name = name
        self.balance = 100
        self.hand = []
        self.stake = 0

    def display_hand(self):
        """Displays the player's hand."""
        hand_str = ', '.join(str(card) for card in self.hand)
        print(f"{self.name}'s hand: {hand_str} (Value: {calculate_hand_value(self.hand)})")

    def calculate_hand_value(hand):
        """Calculates the total value of a hand."""
        value = sum(card.value for card in hand)
        ace_count = sum(1 for card in hand if card.rank == 'A')
        while value > 21 and ace_count:
            value -= 10
            ace_count -= 1
        return value

def blackjack():
    print("Welcome to the Multiplayer Blackjack Game!")

    # Create players
    num_players = int(input("Enter the number of players: "))
    players = [Player(input(f"Enter the name for player {i + 1}: ")) for i in range

```



```

(num_players)]

while any(player.balance > 0 for player in players):
    # Initialize and shuffle the deck
    deck = Deck()
    deck.shuffle()

    # Ask for stakes before dealing cards
    for player in players:
        if player.balance <= 0:
            print(f"{player.name} has no money left to play.")
            continue

        print(f"\n{player.name}'s turn (Balance: ${player.balance})")
        while True:
            try:
                stake = int(input(f"{player.name}, enter your stake for this round: $"))

                if 0 < stake <= player.balance:
                    player.stake = stake
                    break
                else:
                    print(f"Invalid stake amount. You can bet between $1 and ${player.balance}.")
            except ValueError:
                print("Please enter a valid number.")

    # Initial deal
    dealer_hand = [deck.deal_card(), deck.deal_card()]
    for player in players:
        if player.balance > 0:
            player.hand = [deck.deal_card(), deck.deal_card()]

    # Players' turns
    for player in players:
        if player.balance <= 0:
            continue

        player.display_hand()
        print(f"Dealer's hand: {dealer_hand[0]}, ?")

    # Player's decision
    while True:
        choice = input("Do you want to 'hit' (h) or 'stand' (s)? ").strip().lower()

        if choice == 'h':
            player.hand.append(deck.deal_card())
            player.display_hand()
            if calculate_hand_value(player.hand) > 21:
                print(f"{player.name} busts! Dealer wins.")
                player.balance -= player.stake

```

```

        break
    elif choice == 's':
        break
    else:
        print("Invalid choice. Please choose 'h' or 's'.")

# Dealer's turn
print("\nDealer's turn")
while calculate_hand_value(dealer_hand) < 17:
    dealer_hand.append(deck.deal_card())
display_hand("Dealer", dealer_hand)

# Determine winners
dealer_value = calculate_hand_value(dealer_hand)
for player in players:
    if player.balance <= 0:
        continue
    player_value = calculate_hand_value(player.hand)
    if player_value <= 21:
        if dealer_value > 21 or player_value > dealer_value:
            print(f"{player.name} wins!")
            player.balance += player.stake
        elif player_value < dealer_value:
            print(f"{player.name} loses.")
            player.balance -= player.stake
        else:
            print(f"{player.name} ties with the dealer.")
            # Balance remains unchanged

# Check if any player can continue
if not any(player.balance > 0 for player in players):
    print("All players have run out of money. Game over!")
    break

# Ask if the players want to play again
play_again = input("\nDo you want to play another round? (y/n): ").strip().lower()

if play_again != 'y':
    print("Thank you for playing! Goodbye!")
    break

def display_hand(player, hand):
    """Displays the dealer's hand."""
    hand_str = ', '.join(str(card) for card in hand)
    print(f"{player}'s hand: {hand_str} (Value: {calculate_hand_value(hand)})")

if __name__ == "__main__":
    blackjack()

```

Key Changes

1. **Stake Before Dealing:** Players are asked to place their stakes before any cards are dealt. This ensures that betting decisions are made without knowledge of the hands.
2. **Player Class:** Added a **stake** attribute to the **Player** class to track each player's bet for the current round.
3. **Game Flow:** Adjusted the game flow to ensure stakes are placed before hands are displayed.