1. Coding in R (9 points)

Miguel Naranjo, 97943

2023-10-08

Card Game Simulator

Implementation of the card game simulator in R (Fig. 1). Implementation MUST be clearly explained in a PDF document created by using R markdown.

1. **Initialize the Deck**:

   - We start by initializing a deck of cards. In this script, we represent the deck as a vector containing cards with values 2 to 11 (representing number cards and aces) and 10 (representing face cards like Jack, Queen, King). The deck is shuffled to ensure randomness.

```r
# Create a vector of card ranks and suits
ranks <- c("2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace")
suits <- c("Hearts", "Diamonds", "Clubs", "Spades")

# Create an empty data frame to store the cards
deck <- data.frame(Rank = character(0), Suit = character(0))

# Loop through ranks and suits to generate all cards
for (suit in suits) {
  for (rank in ranks) {
    card <- data.frame(Rank = rank, Suit = suit)
    deck <- rbind(deck, card)
  }
}

# Write the deck of cards to a CSV file
write.csv(deck, "poker_deck.csv", row.names = FALSE)
```

2. **Initialize Player and Dealer Hands**:

   - We create empty vectors for the player's and dealer's hands to hold the cards they receive.

```r
# Function to read a CSV file and add a "Value" column to the data frame
read_deck <- function(file_name) {
  # Read the CSV file into a data frame
  deck <- read.csv(file_name, header = TRUE)

  # Define card values
  card_values <- c(
    "2" = 2, "3" = 3, "4" = 4, "5" = 5, "6" = 6, "7" = 7, "8" = 8, "9" = 9, "10" = 10,
    "Jack" = 11, "Queen" = 12, "King" = 13, "Ace" = 14
  )
```

```r
  # Add a "Value" column based on card ranks
  deck$Value <- card_values[deck$Rank]

  return(deck)
}

# Example usage:
file_name <- "poker_deck.csv"  # Replace with the actual CSV file name
deck_data <- read_deck(file_name)

# Print the resulting data frame
print(deck_data)
```

```
##     Rank     Suit Value
## 1      2   Hearts     2
## 2      3   Hearts     3
## 3      4   Hearts     4
## 4      5   Hearts     5
## 5      6   Hearts     6
## 6      7   Hearts     7
## 7      8   Hearts     8
## 8      9   Hearts     9
## 9     10   Hearts    10
## 10  Jack   Hearts    11
## 11 Queen   Hearts    12
## 12  King   Hearts    13
## 13   Ace   Hearts    14
## 14     2 Diamonds     2
## 15     3 Diamonds     3
## 16     4 Diamonds     4
## 17     5 Diamonds     5
## 18     6 Diamonds     6
## 19     7 Diamonds     7
## 20     8 Diamonds     8
## 21     9 Diamonds     9
## 22    10 Diamonds    10
## 23  Jack Diamonds    11
## 24 Queen Diamonds    12
## 25  King Diamonds    13
## 26   Ace Diamonds    14
## 27     2    Clubs     2
## 28     3    Clubs     3
## 29     4    Clubs     4
## 30     5    Clubs     5
## 31     6    Clubs     6
## 32     7    Clubs     7
## 33     8    Clubs     8
## 34     9    Clubs     9
## 35    10    Clubs    10
## 36  Jack    Clubs    11
## 37 Queen    Clubs    12
## 38  King    Clubs    13
## 39   Ace    Clubs    14
## 40     2   Spades     2
```

```
## 41     3   Spades     3
## 42     4   Spades     4
## 43     5   Spades     5
## 44     6   Spades     6
## 45     7   Spades     7
## 46     8   Spades     8
## 47     9   Spades     9
## 48    10   Spades    10
## 49  Jack   Spades    11
## 50 Queen   Spades    12
## 51  King   Spades    13
## 52   Ace   Spades    14
```

3. **Game Loop**:
   - The game runs in a continuous loop (`while (TRUE)`) until the player decides to stop.

```r
# Function to shuffle a deck of cards represented as a data frame
shuffle_deck <- function(deck) {
  # Use the sample function to shuffle the rows of the data frame
  shuffled_deck <- deck[sample(nrow(deck)), ]
  rownames(shuffled_deck) <- NULL  # Reset row names

  return(shuffled_deck)
}

# Example usage:
# Assuming you have a deck data frame, you can shuffle it like this:
shuffled_deck <- shuffle_deck(deck_data)

# Print the shuffled deck
print(shuffled_deck)
```

```
##     Rank     Suit Value
## 1      3   Hearts     3
## 2   Jack   Hearts    11
## 3      5 Diamonds     5
## 4      4   Hearts     4
## 5      5   Spades     5
## 6      5   Hearts     5
## 7      6 Diamonds     6
## 8      8   Spades     8
## 9     10   Hearts    10
## 10     8   Hearts     8
## 11 Queen    Clubs    12
## 12     2    Clubs     2
## 13     5    Clubs     5
## 14 Queen   Hearts    12
## 15     7 Diamonds     7
## 16     4    Clubs     4
## 17 Queen   Spades    12
## 18    10 Diamonds    10
## 19     3   Spades     3
## 20     3 Diamonds     3
## 21     7   Spades     7
## 22   Ace    Clubs    14
```

```
## 23     2 Diamonds     2
## 24 Queen Diamonds    12
## 25     2   Hearts     2
## 26     7   Hearts     7
## 27     6    Clubs     6
## 28     9   Spades     9
## 29  Jack   Spades    11
## 30  King   Spades    13
## 31  Jack Diamonds    11
## 32     8 Diamonds     8
## 33  King    Clubs    13
## 34     9   Hearts     9
## 35    10   Spades    10
## 36     2   Spades     2
## 37   Ace   Spades    14
## 38     9    Clubs     9
## 39  King   Hearts    13
## 40     7    Clubs     7
## 41  Jack    Clubs    11
## 42     4 Diamonds     4
## 43     4   Spades     4
## 44    10    Clubs    10
## 45     8    Clubs     8
## 46     6   Hearts     6
## 47     3    Clubs     3
## 48     9 Diamonds     9
## 49   Ace   Hearts    14
## 50   Ace Diamonds    14
## 51     6   Spades     6
## 52  King Diamonds    13
```

4. **Player's Turn**:
   - During the player's turn, the script displays the player's hand and one of the dealer's cards.
   - The player is asked to choose between "hit" (take another card) or "stand" (stop taking cards).
   - If the player chooses to hit, a card is dealt from the deck and added to the player's hand. The hand value is calculated.
   - If the player chooses to stand, the player's turn ends.

```r
# Function to deal n cards from a shuffled deck, ensuring no card is dealt twice
deal_cards <- function(deck, n) {
  # Initialize an empty list to keep track of dealt cards
  dealt_cards <- list()

  # Initialize an empty data frame to store the dealt cards
  dealt_deck <- data.frame(Rank = character(0), Suit = character(0), Value = numeric(0))

  # Check if there are enough cards left to deal
  if (n > nrow(deck)) {
    stop("Not enough cards left in the deck to deal ", n, " cards.")
  }

  for (i in 1:n) {
    # Check if all cards have been dealt
```

```
    if (nrow(deck) == 0) {
      warning("All cards have been dealt.")
      break
    }

    # Randomly select a card from the remaining deck
    random_index <- sample(nrow(deck), 1)
    dealt_card <- deck[random_index, ]

    # Remove the dealt card from the deck
    deck <- deck[-random_index, , drop = FALSE]

    # Add the dealt card to the dealt cards list and data frame
    dealt_cards[[i]] <- dealt_card
    dealt_deck <- rbind(dealt_deck, dealt_card)
  }

  return(dealt_deck)
}

# Example usage:
# Assuming you have a shuffled deck represented by the 'shuffled_deck' data frame, you can dea
dealt_cards <- deal_cards(shuffled_deck, n = 5)

# Print the dealt cards
print(dealt_cards)
```

```
##    Rank     Suit Value
## 9    10   Hearts    10
## 52 King Diamonds    13
## 15    7 Diamonds     7
## 28    9   Spades     9
## 32    8 Diamonds     8
```

5. **Check for Player Bust or Blackjack**:

   - After each player's action, we check if the player's hand value exceeds 21 (bust) or equals 21 (blackjack). In case of a bust or blackjack, the game round ends.

```
# Function to calculate the total value of a hand
calculate_hand_value <- function(hand) {
  hand_value <- sum(hand)

  # Check for aces and adjust for soft hands
  if (any(hand == 11) && hand_value > 21) {
    num_aces <- sum(hand == 11)
    while (num_aces > 0 && hand_value > 21) {
      hand_value <- hand_value - 10
      num_aces <- num_aces - 1
    }
  }

  return(hand_value)
}
```

```r
# Function to deal a new card
deal_card <- function(deck, hand) {
  card <- deck[1]
  deck <- deck[-1]
  hand <- c(hand, card)
  return(list(deck, hand))
}

# Initialize the deck
deck <- c(2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11)
deck <- rep(deck, 4)

# Shuffle the deck
deck <- sample(deck)

# Initialize player and dealer hands
player_hand <- numeric(0)
dealer_hand <- numeric(0)

# Deal the initial two cards to the player and dealer
deck_and_player <- deal_card(deck, player_hand)
deck <- deck_and_player[[1]]
player_hand <- deck_and_player[[2]]

deck_and_dealer <- deal_card(deck, dealer_hand)
deck <- deck_and_dealer[[1]]
dealer_hand <- deck_and_dealer[[2]]

# Game loop
while (TRUE) {
  # Calculate hand values
  player_value <- calculate_hand_value(player_hand)
  dealer_value <- calculate_hand_value(dealer_hand)

  # Display hands
  cat("Your hand:", paste(player_hand, collapse = ", "), "\n")
  cat("Dealer's hand:", paste(dealer_hand[1], "??"), "\n")

  # Check for blackjack
  if (player_value == 21) {
    cat("Blackjack! You win!\n")
    break
  }

  # Player's turn
  action <- readline(prompt = "Do you want to hit or stand? (h/s): ")

  if (action == "h") {
    deck_and_player <- deal_card(deck, player_hand)
    deck <- deck_and_player[[1]]
    player_hand <- deck_and_player[[2]]
  } else if (action == "s") {
    break
```

```
  }

  # Check for player bust
  if (player_value > 21) {
    cat("Bust! You lose!\n")
    break
  }

  # Dealer's turn
  while (dealer_value < 17) {
    deck_and_dealer <- deal_card(deck, dealer_hand)
    deck <- deck_and_dealer[[1]]
    dealer_hand <- deck_and_dealer[[2]]
    dealer_value <- calculate_hand_value(dealer_hand)
  }

  # Check for dealer bust or compare hands
  if (dealer_value > 21) {
    cat("Dealer busts! You win!\n")
  } else if (dealer_value == player_value) {
    cat("It's a tie! Push.\n")
  } else if (dealer_value > player_value) {
    cat("Dealer wins!\n")
  } else {
    cat("You win!\n")
  }

  break
}
```

```
## Your hand: 10
## Dealer's hand: 2 ??
## Do you want to hit or stand? (h/s):
## Dealer wins!
```

```
cat("Thanks for playing!\n")
```

```
## Thanks for playing!
```

6. **Dealer's Turn**:
   - The dealer's turn begins. The dealer will draw cards until their hand value reaches 17 or higher.
   - Cards are dealt to the dealer from the deck, and the hand value is calculated after each card.

```
# Function to calculate the total value of a hand
calculate_hand_value <- function(hand) {
  hand_value <- sum(hand)

  # Check for aces and adjust for soft hands
  if (any(hand == 11) && hand_value > 21) {
    num_aces <- sum(hand == 11)
    while (num_aces > 0 && hand_value > 21) {
      hand_value <- hand_value - 10
      num_aces <- num_aces - 1
    }
```

```
  }

  return(hand_value)
}

# Function to deal a new card
deal_card <- function(deck, hand) {
  card <- deck[1]
  deck <- deck[-1]
  hand <- c(hand, card)
  return(list(deck, hand))
}

# Initialize the deck
deck <- c(2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11)
deck <- rep(deck, 4)

# Shuffle the deck
deck <- sample(deck)

# Initialize player and dealer hands
player_hand <- numeric(0)
dealer_hand <- numeric(0)

# Initialize variables to track wins and losses
wins <- 0
losses <- 0

# Game loop
while (TRUE) {
  # Calculate hand values
  player_value <- calculate_hand_value(player_hand)
  dealer_value <- calculate_hand_value(dealer_hand)

  # Display hands
  cat("Your hand:", paste(player_hand, collapse = ", "), "\n")
  cat("Dealer's hand:", paste(dealer_hand[1], "??"), "\n")

  # Check for blackjack
  if (player_value == 21) {
    cat("Blackjack! You win!\n")
    wins <- wins + 1
    break
  }

  # Player's turn
  action <- readline(prompt = "Do you want to hit or stand? (h/s): ")

  if (action == "h") {
    deck_and_player <- deal_card(deck, player_hand)
    deck <- deck_and_player[[1]]
    player_hand <- deck_and_player[[2]]
  } else if (action == "s") {
```

```
    break
  }

  # Check for player bust
  if (player_value > 21) {
    cat("Bust! You lose!\n")
    losses <- losses + 1
    break
  }

  # Dealer's turn
  while (dealer_value < 17) {
    deck_and_dealer <- deal_card(deck, dealer_hand)
    deck <- deck_and_dealer[[1]]
    dealer_hand <- deck_and_dealer[[2]]
    dealer_value <- calculate_hand_value(dealer_hand)
  }

  # Check for dealer bust or compare hands
  if (dealer_value > 21) {
    cat("Dealer busts! You win!\n")
    wins <- wins + 1
  } else if (dealer_value == player_value) {
    cat("It's a tie! Push.\n")
  } else if (dealer_value > player_value) {
    cat("Dealer wins!\n")
    losses <- losses + 1
  } else {
    cat("You win!\n")
    wins <- wins + 1
  }

  # Ask the user if they want to continue playing
  continue_playing <- readline(prompt = "Do you want to continue playing? (Y/N): ")
  if (toupper(continue_playing) != "Y") {
    break
  }

  # Reset hands for the next round
  player_hand <- numeric(0)
  dealer_hand <- numeric(0)
}
```

```
## Your hand:
## Dealer's hand: NA ??
## Do you want to hit or stand? (h/s):
## Dealer wins!
## Do you want to continue playing? (Y/N):
```

```
# Save the results to a CSV file
results <- data.frame(Wins = wins, Losses = losses)
write.csv(results, "blackjack_results.csv", row.names = FALSE)

cat("Thanks for playing!\n")
```

```
## Thanks for playing!
```

7. **Check for Dealer Bust or Compare Hands**:
   - After the dealer's turn, we check if the dealer's hand value exceeds 21 (bust).
   - If the dealer busts, the player wins.
   - If not, we compare the final hand values of the player and the dealer to determine the winner.
8. **Recording Wins and Losses**:
   - We keep track of the number of wins and losses for the player throughout the game using the `wins` and `losses` variables.
9. **Ask to Continue Playing**:
   - After each round, the player is asked if they want to continue playing. If the player enters "Y" (yes), a new round begins. If the player enters "N" (no), the game loop exits.
10. **Saving Results**:
    - At the end of the game, the script creates a data frame (`results`) to store the number of wins and losses.
    - It uses the `write.csv` function to save these results in a CSV file named "blackjack_results.csv."
11. **Exit the Game**:
    - The script thanks the player for playing and exits.