Project Documentation

Description

The project is a text-based implementation of the classic casino card game Blackjack. In this project, the player can interact with the game by choosing to hit or stand and compete against a computerized dealer. The goal is to have a hand value higher than the dealer and as close to 21 as possible without going over (busting).

Solution Principle

The project utilizes core Python elements, including functions, data structures (lists, dictionaries), control flow and file handling. It employs a modular design to enhance code readability, maintainability and ease of extension.

Project Structure

The project is structured into functions, each responsible for specific parts of the game. Key functions include:

- `create_deck()`: Generates a standard deck of playing cards.
- `shuffle_deck(deck)`: Shuffles the deck randomly.
- `deal_card(deck)`: Deals a card from the deck, creating a new deck if necessary.
- `first_hand(hand)`: Displays the player's and dealer's initial two cards, hiding the dealer’s second card.
- `display_hand(hand, show_all=True)`: Displays the player's or dealer's hand with Unicode symbols for suits.
- `calculate_hand(hand)`: Determines the total value of a hand, considering special rules for Aces.
- `welcome()`: Displays the welcome message with general instructions.
- `play_game()`: Initiates the game and accepts player decisions.
- `play_again()`: Asks the user if they want to play again or end the game.
- `load_game_stats()`: Loads previous game statistics from a JSON file.
- `save_game_stats(stats)`: Saves the current game statistics to the JSON file (cumulative).
- `display_game_stats(stats)`: Displays the game statistics.

Functions and Their Interrelationships

- Main Function
  - `main()`
  - Calls the `welcome()` function to display the welcome message and game instructions.
  - Manages the player's turn and dealer's actions by using loops and calling the game logic functions.
  - Tracks and updates game statistics.
  - Prompts the player to play again or quit at the end of each round.
- Game Logic Functions
  - `create_deck()`
  - `shuffle_deck(deck)`
  - `deal_card(deck)`
  - `first_hand(hand)`
  - `display_hand(hand, show_all=True)`
  - `calculate_hand(hand)`
  - `play_again()`
- Game Stats Functions
  - `load_game_stats()`
  - `save_game_stats(stats)`
  - `display_game_stats(stats)`
  - Handle loading, saving, updating, and displaying game statistics.

External Libraries

The project employs the built-in `json` library for handling file actions, specifically for saving and loading game statistics.

The `random` module is used to shuffle the deck of cards, introducing an element of randomness to the order in which the cards are dealt.

The `namedtuple()` function is used to represent individual playing cards. The `Card` named tuple is used to create instances of playing cards when generating the deck of cards. Each playing card in the deck is represented as an instance of this named tuple, where `suit` represents the suit of the card and `rank` represents the rank of the card. This allows access to fields using attribute names (`card.suit`, `card.rank`) instead of numerical indices. Named tuples are immutable, making them suitable for representing playing cards, which have set values.

Game Logic

The game starts by calling the `welcome()` function and reading the ‘`welcome.txt`’ file, which shows a welcome message and general instructions about how to play the game. If the ‘`welcome.txt`’ file is not found, the program will display an error message and proceed with the game.

The program will also display simple statistics of previously played games, including the total number of games and the results (wins, losses, ties and busts). The game statistics are read from the ‘`game_stats.json`’ file. if the file does not exist, the `load_game_stats()` function returns a default set of statistics with initial values (all 0).

The goal of the game is to beat the dealer by having a hand value higher than the dealer and as close to 21 without going over (busting). Initially, the computer deals two cards to both the player and the dealer. The second card of the dealer’s hand is hidden (until the dealer’s turn or if the player busts).

The player’s turn is first. The player can choose to hit (take a card) or stand (keep the current hand). The desired action is indicated by entering the corresponding key: `[H]` to hit or `[S]` to stand. The game logic implements a simple loop to prevent invalid output. If the player busts, the dealer wins automatically. The dealer’s hand is not played further, but the initial two-card hand is revealed.

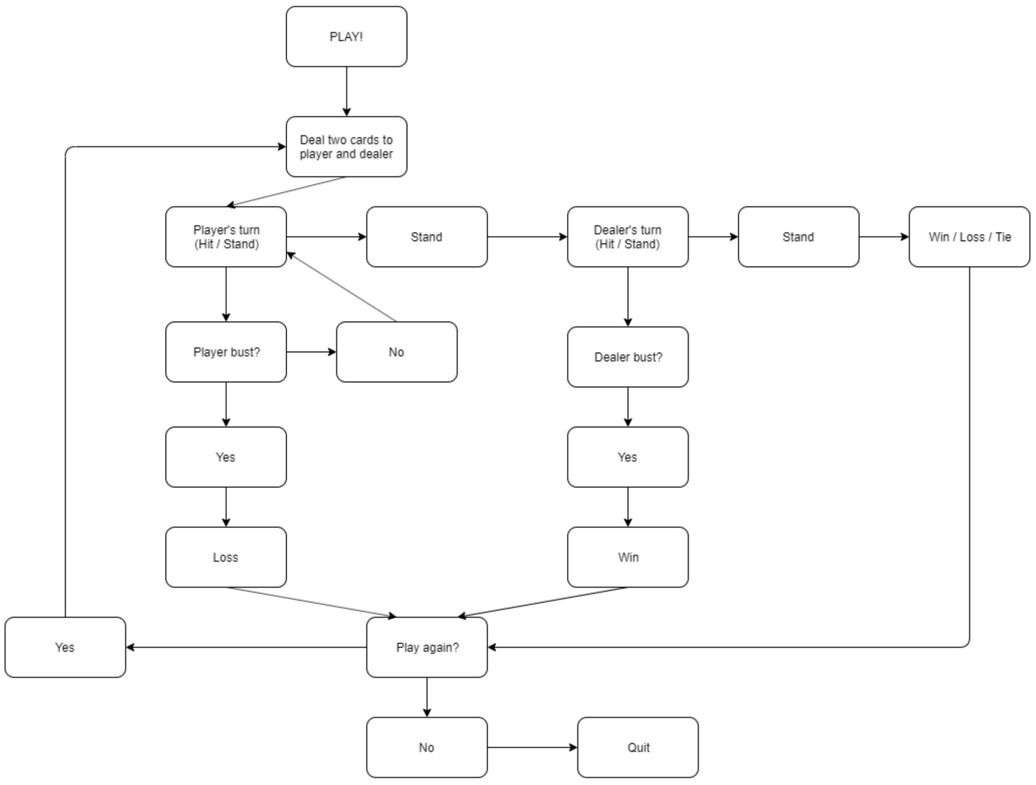
After the player stands, it is the dealer’s turn. The dealer will continue to take a card until the hand value is 17 or more. If the dealer busts, the player wins automatically. Otherwise, the hand values are compared and a result (win, loss or tie) is determined.

The result is recorded in the game statistics file in JSON format.

Finally, the player is asked if they want to play again or quit the game. Similar to the card actions, the desired action is selected by entering the corresponding key: `[P]` to play again or `[Q]` to quit the game.

If the player chooses to play again, the game logic runs again starting from dealing new cards to the player and the dealer. The player can play as many rounds as they want. The `deal_card()` function will create a new deck of cards if the current deck has no more cards left.

The following flowchart depicts the progression of the game.



How to Run

Perform the following steps:

1. Download or pull the files from the GitHub repository: https://github.com/dotonebit/blackjack
2. Ensure that Python 3 is installed on the system.
3. Execute the '`blackjack.py`' script in a Python environment.
    - The '`welcome.txt`' and '`game_stats.json`' files are needed for the full functionality of the game, but the game will also run without them.
    - If the files are not found, the program will proceed with the game anyway or show default values and create the missing file.
4. Follow the on-screen instructions to play the game.

Conclusion

The Blackjack game written in Python offers an engaging and interactive implementation of the classic card game, providing the player with a straightforward console-based user experience. The code structure allows for easy extension and modification, making it a suitable project for learning and experimentation.