Blackjack Project

A simple Blackjack game built with Pygame in Python. The project is separated into 3 files, each having its own purpose

- **constants.py** – All constants (screen size, colors, suits, ranks, etc.) and functions for loading images.
- **blackjack_setup.py** – Contains the logic behind BlackJack (card, deck, player, dealer, and main game classes).
- **main.py** – The UI of game, event handling and main game loop.

Features

1. **Menu State**: Allows adjusting the bet with "+" and "–" buttons, then deals cards.
2. **Playing State**: Includes "Hit" and "Stand" buttons for player actions. Dealer automatically hits until a 17 or higher is reached.
3. **Game Over State**: Shows the outcome (win, lose, draw, or blackjack) and final scores, and offers a "Play Again" button to restart.
4. **Balance Tracking**: Each hand's outcome updates the player's balance.
5. **Card Rendering**: Card images are displayed, including a hidden first dealer card during the "Playing" state.

Balance distribution

This game starts the player with a **default balance** of 1,000 (configurable in BlackjackGame or Player). Betting and payouts work as follows:

1. **Minimum Bet**:
    - The minimum bet is **$10**. The "Minus" button won't work if user tries to put a lower bet
2. **Placing a Bet**:
    - When the player clicks "Deal" in the **Menu** state, the chosen bet is subtracted from the player's balance.
3. **Round Outcomes**:
    - **Lose**: The player loses their bet; nothing is returned.
    - **Win**: The player's balance increases by **2× the bet** (original bet + equal winnings).
    - **Draw**: The full bet is returned.
    - **Blackjack**: A two-card 21 pays **3:2**.
4. **Balance After Round**:
    - Once the round concludes (Game Over state), the updated balance is displayed.

- The player can click “Play Again” to return to the menu, place another bet, and continue playing with the new balance.

Files quick through

1. constants.py

- **Purpose**:
  - Stores core constants such as screen dimensions, color tuples, game states, suits, and ranks.
  - Provides **load_background()** and **load_card()** functions to load and scale images.
- **Key Items**:
  - SCREEN_WIDTH, SCREEN_HEIGHT, FPS, color triples (WHITE, BLACK, GREEN, etc.).
  - STATE_MENU, STATE_PLAYING, STATE_GAME_OVER.
  - SUITS, RANKS arrays for generating card data.
  - A fallback strategy if an image is missing, using a solid-colored Surface.

2. blackjack_setup.py

- **Purpose**:
  1. Implements the **model** of the game (Card, Deck, Hand, Player, Dealer) plus a BlackjackGame class.
  2. Encapsulates dealing, betting, win/loss logic, and game state transitions.
- **Key Classes**:
  1. **Card** – Holds a suit, rank, and computed value (Aces default to 11).
  2. **Deck** – Manages a shuffled list of 52 Card objects, deals cards from the top.
  3. **Hand** – Maintains a list of Cards, calculates a score (handling Aces as 1 or 11), checks busts/blackjacks.
  4. **Player** – Tracks balance, bet, and a Hand. Has methods for betting, winning, losing, pushing (draw).
  5. **Dealer** – Has a Hand and a should_hit() method to automate drawing until reaching 17+.
  6. **BlackjackGame** – Orchestrates a round of Blackjack:
     - **start_game(bet_amount)** – Places a bet, deals initial cards, checks immediate blackjack.
     - **player_hit()** – Player draws a card, checks for bust.
     - **player_stand()** – Dealer draws if needed, then calls determine_result().

        - **determine_result()** – Compares final scores, updates outcome, and transitions to GAME_OVER.
        - **reset_game()** – Resets deck/hands for a new round, returns to the menu.

3. main.py

- **Purpose**:
    - Handles the main event loop and rendering for different game states (menu, playing, game over).
- **Key Functions**:
    - **main_loop()** – Runs the main Pygame loop, updating the screen at FPS and responding to user inputs.
    - **draw_menu_state(...)**, **draw_playing_state(...)**, **draw_game_over_state(...)** – Render the game background, cards, buttons, and text based on the current state.
    - **handle_menu_events(...)**, **handle_playing_events(...)**, **handle_game_over_events(...)** – Process mouse clicks for betting, dealing, hitting, standing, or resetting.
- **User Interaction**:
    - **Menu**: Click “+” or “–” to adjust the bet, then “Deal” to start.
    - **Playing**: Click “Hit” to draw a card, or “Stand” to let the dealer finish.
    - **Game Over**: Displays final hands and outcome, with “Play Again” to restart.