```
#!/usr/bin/perl
# Enhanced Blackjack Game – Session-based bankroll tracking with restart option
# Date: September 2025

use strict;
use warnings;
use 5.010;
use List::Util qw(shuffle);
use File::Path qw(make_path);
use Scalar::Util qw(looks_like_number);

# Game constants
use constant {
    STARTING_BANKROLL   => 1000,
    MIN_BET             => 1,
    MAX_BET_RATIO       => 1.0,    # Can bet up to 100% of bankroll
    BLACKJACK_PAYOUT    => 1.5,
    DEALER_STAND        => 17,
    RESHUFFLE_THRESHOLD => 15,
    DATA_DIR            => "blackjack_data"
};

# Game state variables
my %game = (
    player_name      => "",
    bankroll         => 0,
```

```
    starting_bankroll => 0,    # NEW: bankroll at start of current session
    bet             => 0,
    hands_played    => 0,
    hands_won       => 0,
    hands_lost      => 0,
    hands_tied      => 0,
    total_winnings  => 0,
    deck            => [],
    player_cards    => [],
    dealer_cards    => [],
    player_total    => 0,
    dealer_total    => 0
);

# Base deck
my @base_deck = (
    ('A') x 4, ('2') x 4, ('3') x 4, ('4') x 4, ('5') x 4,
    ('6') x 4, ('7') x 4, ('8') x 4, ('9') x 4, ('10') x 4,
    ('J') x 4, ('Q') x 4, ('K') x 4
);

#---------------- MAIN ----------------#
sub main {
    print_welcome_banner();
    get_player_name();
    initialize_game();
```

```
    while (1) {
        last unless play_hand();
        last unless ask_play_again();
    }

    print_farewell();
}

#---------------- DISPLAY ----------------#
sub print_welcome_banner {
    print "=" x 50, "\n";
    print "           ENHANCED BLACKJACK GAME\n";
    print "=" x 50, "\n";
    print "Goal: Get as close to 21 as possible without going over!\n";
    print "Dealer stands on 17\n\n";
}

sub print_farewell {
    print "\n", "=" x 40, "\n";
    print "Thanks for playing Enhanced Blackjack, $game{player_name}!\n";
    print_final_stats();
    print "=" x 40, "\n";
}

sub display_initial_hands {
```

```
    print "Your cards: ", join(' ', @{$game{player_cards}}),
        " (Total: $game{player_total})\n";
    print "Dealer shows: [HIDDEN] $game{dealer_cards}[1]\n\n";
}

sub display_all_hands {
    print "Your cards: ", join(' ', @{$game{player_cards}}),
        " (Total: $game{player_total})\n";
    print "Dealer cards: ", join(' ', @{$game{dealer_cards}}),
        " (Total: $game{dealer_total})\n";
}

#---------------- INIT & SAVE ----------------#
sub get_player_name {
    print "What's your name, player? ";
    chomp(my $name = <STDIN>);
    $name =~ s/^\s+|\s+$//g;
    $name = ucfirst(lc($name)) if $name;
    if ($name eq "" || length($name) < 2) {
        $name = "Player";
        print "I'll just call you Player then!\n";
    }
    $game{player_name} = $name;
    print "\nNice to meet you, $game{player_name}!\n\n";
}
```

```
sub initialize_game {
    setup_data_directory();
    load_game_state();
    shuffle_new_deck();

    if ($game{hands_played} > 0) {
        print "Welcome back, $game{player_name}! Loading your previous game...\n";
        print_current_stats();
    } else {
        print "Starting fresh with \$", STARTING_BANKROLL, ", $game{player_name}!\n";
    }
    print "\n";

    # Set session starting bankroll now
    $game{starting_bankroll} = $game{bankroll};
}

sub setup_data_directory {
    unless (-d DATA_DIR) {
        make_path(DATA_DIR) or die "Cannot create data directory: $!";
    }
}

sub get_player_filename {
    my $safe_name = $game{player_name};
    $safe_name =~ s/[^a-zA-Z0-9_-]//g;
```

```
    $safe_name = lc($safe_name);
    return DATA_DIR . "/" . $safe_name . "_data.txt";
}

sub load_game_state {
    my $player_file = get_player_filename();
    if (-e $player_file) {
        open my $fh, '<', $player_file or die "Cannot read player file: $!";
        while (my $line = <$fh>) {
            chomp $line;
            my ($key, $value) = split /=/, $line, 2;
            $game{$key} = looks_like_number($value) ? $value : $value if defined $value;
        }
        close $fh;
    } else {
        # New player defaults
        $game{bankroll} = STARTING_BANKROLL;
        $game{hands_played} = 0;
        $game{hands_won}    = 0;
        $game{hands_lost}   = 0;
        $game{hands_tied}   = 0;
        $game{total_winnings} = 0;
    }
    $game{bankroll} = STARTING_BANKROLL if $game{bankroll} < 0;
}
```

```
sub save_game_state {
    my $player_file = get_player_filename();
    open my $fh, '>', $player_file or die "Cannot write player file: $!";
    for my $key (qw(player_name bankroll hands_played hands_won hands_lost hands_tied
total_winnings)) {
        print $fh "$key=$game{$key}\n";
    }
    close $fh;
}

#---------------- GAME FLOW ----------------#
sub shuffle_new_deck {
    @{$game{deck}} = shuffle @base_deck;
    print "Shuffling a fresh deck...\n" if $game{hands_played} > 0;
}

sub play_hand {
    return 0 unless check_bankroll();
    get_bet() or return 0;
    deal_initial_cards();

    # Natural blackjack
    if ($game{player_total} == 21) {
        handle_blackjack();
        return 1;
    }
```

```
    return 0 unless player_turn();
    dealer_turn() unless $game{player_total} > 21;
    determine_winner();
    save_game_state();
    return 1;
}

#---------------- BETTING & MONEY ----------------#
sub check_bankroll {
    if ($game{bankroll} < MIN_BET) {
        print "You're out of money! You need at least \$", MIN_BET, " to play.\n";
        print "Would you like to restart with a fresh \$", STARTING_BANKROLL, "? (y/n): ";
        chomp(my $ans = <STDIN>);
        if (lc($ans) eq 'y') {
            $game{bankroll}         = STARTING_BANKROLL;
            $game{starting_bankroll} = STARTING_BANKROLL;
            $game{hands_played}     = 0;
            $game{hands_won}        = 0;
            $game{hands_lost}       = 0;
            $game{hands_tied}       = 0;
            $game{total_winnings}   = 0;
            save_game_state();
            print "\nBankroll reset to \$$game{bankroll}. Let's keep playing!\n\n";
            return 1;
        }
```

```
        return 0;
    }
    return 1;
}

sub get_bet {
    my $max_bet = int($game{bankroll} * MAX_BET_RATIO);
    print "Current bankroll: \$$game{bankroll}\n";
    print "Minimum bet: \$", MIN_BET, ", Maximum bet: \$$max_bet\n";
    while (1) {
        print "How much would you like to bet? (\$", MIN_BET, "-\$${max_bet}) or 'quit': \$";
        chomp(my $input = <STDIN>);
        return 0 if $input =~ /^q(uit)?$/i;
        unless (looks_like_number($input)) {
            print "Please enter a valid number.\n"; next;
        }
        $game{bet} = int($input);
        if    ($game{bet} < MIN_BET) { print "Minimum bet is \$", MIN_BET, ".\n"; }
        elsif ($game{bet} > $max_bet){ print "You can't bet more than \$$max_bet.\n"; }
        else {
            print "Bet accepted: \$$game{bet}\n\n";
            return 1;
        }
    }
}
```

```
#---------------- DEALING ----------------#
sub deal_initial_cards {
    @{$game{player_cards}} = ();
    @{$game{dealer_cards}} = ();
    shuffle_new_deck() if @{$game{deck}} < RESHUFFLE_THRESHOLD;

    push @{$game{player_cards}}, shift @{$game{deck}};
    push @{$game{dealer_cards}}, shift @{$game{deck}};
    push @{$game{player_cards}}, shift @{$game{deck}};
    push @{$game{dealer_cards}}, shift @{$game{deck}};

    calculate_totals();
    display_initial_hands();
}

sub calculate_totals {
    $game{player_total} = calculate_hand_total(@{$game{player_cards}});
    $game{dealer_total} = calculate_hand_total(@{$game{dealer_cards}});
}

sub calculate_hand_total {
    my @cards = @_;
    my $total = 0; my $aces = 0;
    for my $card (@cards) {
        if    ($card =~ /^[2-9]$|^10$/) { $total += $card; }
        elsif ($card =~ /^[JQK]$/)      { $total += 10;    }
```

```
        elsif ($card eq 'A')          { $aces++;       }
    }
    for (1..$aces) {
        $total += ($total + 11 <= 21) ? 11 : 1;
    }
    return $total;
}

#---------------- PLAYER/DEALER ACTIONS ----------------#
sub handle_blackjack {
    print "BLACKJACK! You got 21!\n";
    my $winnings = int($game{bet} * BLACKJACK_PAYOUT);
    $game{bankroll} += $winnings;
    $game{hands_won}++;
    $game{total_winnings} += $winnings;
    print "You win \$$winnings! Your bankroll is now \$$game{bankroll}.\n\n";
}

sub player_turn {
    while ($game{player_total} < 21) {
        print "Your total is $game{player_total}. (h)it or (s)tand? ";
        chomp(my $choice = lc(<STDIN>));
        if ($choice eq 'h' || $choice eq 'hit') {
            hit_player();
            if ($game{player_total} > 21) {
                print "BUST! You went over 21.\n";
```

```
            return 1;
        }
    } elsif ($choice eq 's' || $choice eq 'stand') {
        print "You stand with $game{player_total}.\n\n";
        return 1;
    } else {
        print "Please enter 'h' or 's'.\n";
    }
  }
  return 1;
}

sub hit_player {
  my $card = shift @{$game{deck}};
  push @{$game{player_cards}}, $card;
  calculate_totals();
  print "You drew: $card\n";
  print "Your cards: ", join(' ', @{$game{player_cards}}),
      " (Total: $game{player_total})\n\n";
}

sub dealer_turn {
  print "Dealer reveals hidden card: $game{dealer_cards}[0]\n";
  display_all_hands(); print "\n";
  while ($game{dealer_total} < DEALER_STAND) {
    print "Dealer hits...\n";
```

```
        my $card = shift @{$game{deck}};
        push @{$game{dealer_cards}}, $card;
        calculate_totals();
        print "Dealer drew: $card (Total: $game{dealer_total})\n";
    }
    if ($game{dealer_total} <= 21) {
        print "Dealer stands with $game{dealer_total}.\n";
    } else {
        print "Dealer BUSTS with $game{dealer_total}!\n";
    }
    print "\n";
}

#---------------- RESULTS ----------------#
sub determine_winner {
    $game{hands_played}++;
    display_all_hands(); print "\n";

    my $winnings = 0;
    if    ($game{player_total} > 21)       { $winnings = -$game{bet}; $game{hands_lost}++; print "You BUSTED!\n"; }
    elsif ($game{dealer_total} > 21)       { $winnings =  $game{bet}; $game{hands_won}++;  print "Dealer BUSTED! You WIN!\n"; }
    elsif ($game{player_total} > $game{dealer_total}) { $winnings = $game{bet}; $game{hands_won}++; print "You WIN!\n"; }
    elsif ($game{player_total} < $game{dealer_total}) { $winnings = -$game{bet}; $game{hands_lost}++; print "You LOSE!\n"; }
```

```
    else                        { $game{hands_tied}++; print "PUSH (Tie)!\n"; }

    $game{bankroll} += $winnings;
    $game{total_winnings} += $winnings;

    if    ($winnings > 0) { print "You win \$$winnings!\n"; }
    elsif ($winnings < 0) { print "You lose \$", abs($winnings), "!\n"; }
    else                  { print "Your bet is returned.\n"; }

    print "Your bankroll is now: \$$game{bankroll}\n\n";
}

sub ask_play_again {
    return 0 unless check_bankroll();
    print "Play another hand, $game{player_name}? (y/n/stats): ";
    chomp(my $choice = lc(<STDIN>));
    if ($choice eq 'y' || $choice eq 'yes') {
        print "\n" . "=" x 30 . " NEW HAND " . "=" x 30 . "\n\n";
        return 1;
    } elsif ($choice eq 'stats') {
        print_current_stats();
        return ask_play_again();
    } else {
        return 0;
    }
}
```

```
sub print_current_stats {
    return unless $game{hands_played} > 0;
    print "\nGAME STATISTICS:\n", "=" x 30, "\n";
    print "Hands played: $game{hands_played}\n";
    print "Hands won:    $game{hands_won}\n";
    print "Hands lost:   $game{hands_lost}\n";
    print "Hands tied:   $game{hands_tied}\n";
    my $win_pct = sprintf("%.1f", ($game{hands_won} / $game{hands_played}) * 100);
    print "Win percentage: $win_pct%\n";
    if ($game{total_winnings} >= 0) {
        print "Total winnings: +\$$game{total_winnings}\n";
    } else {
        print "Total winnings: -\$", abs($game{total_winnings}), "\n";
    }
    print "Current bankroll: \$$game{bankroll}\n";
    print "=" x 30, "\n\n";
}

sub print_final_stats {
    print_current_stats();
    my $diff = $game{bankroll} - $game{starting_bankroll};
    if ($diff > 0) {
        print "You made a profit of \$$diff this session!\n";
    } elsif ($diff < 0) {
        print "You lost \$", abs($diff), " this session.\n";
```

```
    } else {
        print "You broke even this session.\n";
    }
}

#---------------- START ----------------#
main();
```