Advanced Blackjack Strategy - Programmer Documentation

Connor Finch (cmf), 5-31-2021

Table of Contents

Python Source Files **2**

basic_strategy.py 2

betting_system.py 2

card_count.py 3

card.py 3

const.py 3

deck.py 3

game.py 4

hand.py 4

menu.py 5

minigame.py 5

participant.py 6

visualization.py 6

Python Source Files

basic_strategy.py

Uses the value of the player's hand and dealer's up card to read from one of three dictionaries defined in const.py depending on the cards in the player's hand.
Functions:

1. basic_strategy()
    a. Takes in the player's hand and dealer's card and returns the move suggested by the basic strategy dictionaries.

betting_system.py

Defines the utilities required for the system to properly manage the player's balance and bets that they make throughout a blackjack game.
Functions:

1. getBet()
    a. Called at the start of every new round of blackjack in order to retrieve the amount of money that the player is willing to bet.
2. validBet()
    a. Checks to see if the player's bet is greater than zero and doesn't cause their balance to go into the negatives.
3. double()
    a. Updates the player's balance and betsize (the money a user has betted) according to the blackjack.
4. validBalance()
    a. Checks to see if the player's balance is less than zero.
5. won()
    a. Called if the player beats the dealer. The player's balance is updated accordingly.
6. tie()
    a. Called if the player and dealer tie. The player's balance remains the same as when they started the round.
7. blackjack()
    a. Called if the player gets a blackjack. The player is paid 3:2 since their hand was a blackjack.

card_count.py

This file defines the CardCounting class which keeps track of the running and true counts.
Functions:

1. runningCount()
    a. Takes in a card object and then updates the running count based on the card counting method currently in use.
2. trueCount()
    a. Takes in the approximate number of decks that are still being played with and divides the running count by this number.
3. displayInfo()
    a. Provides an explanation on how to interpret the true count.

card.py

This file defines the Card class in order to keep track of the rank, suit, and value of a card.
Functions:

1. __eq__()
    a. Implemented in order to provide a way to easily compare Card objects.

const.py

This file defines all the constant data that will be used throughout the program; therefore, it has no functions implemented in it.

deck.py

This file defines the Deck class which is a list of the Card data type defined in card.py.
Functions:

1. shuffle()
    a. Used when the deck is created, so the cards in the deck are in a random order.
2. deal()
    a. Removes a card from the deck and returns that card to the function that called it.
3. reshuffle()

    a. Returns true or false depending on whether the deck needs to be replaced with a new one. This function returns true when over 75% of the cards in the deck have been played; otherwise, it will return false.

game.py

This file defines the Game class which is responsible for handling what goes on at the start and end of each round of blackjack.
Functions:

1. playGame()
    a. Deals the starting hands to the player and dealer. It then calls the playHands() function defined in the Player class which will handle all of the player's input.
2. compareHands()
    a. Calls the compare() function for every hand that the player has. This is required because a user might have more than one hand if they chose to split.
3. compare()
    a. Compares the values of the player's and dealer's hand and outputs the appropriate information based on whether the player's hand won, tied, or lost to the dealer's.

hand.py

This file defines the Hand class which implements rules of blackjack as functions.
Functions:

1. addCard()
    a. Takes in a card object and appends it to a list of cards which are stored as a member of the Hand object.
2. getValue()
    a. Computes and updates the value of the hand and then returns the value to the calling function.
3. hittable()
    a. Checks to see hitting is a valid move for the hand.
4. busted()
    a. Checks to see if the value of the hand is greater than 21.
5. blackjack()
    a. Checks to see if the initial hand value is 21.
6. doublable()
    a. Checks to see doubling is a valid move for the hand.
7. splittable()

a. Checks to see if splitting is a valid move for the hand.

menu.py

This file defines the Menu class which is responsible for taking in player input and starting different game modes. This is file used to start the program which is done by typing “python ./menu.py” into a terminal.

Functions

1. again()
    a. Takes in an input from the player and determines if they want to play again.
2. change_shoe_size()
    a. Takes in an input from the player and changes the number of decks being played with based on the inputted number.
3. home()
    a. Provides four different options for the player to select from. These options let the user start playing blackjack games, basic strategy practice/minigame, change the number of decks being played with, or exit the program.
4. welcome_message()
    a. Prints out “Welcome to the Advanced Blackjack System” whenever the player accesses the main menu.

minigame.py

This file defines logic needed to properly run the minigame such as the comparisons between the player’s hand and dealer’s up card.

Functions:

1. again()
    a. Exactly the same as the function described in menu.py. The reason why it is also needed in this file is because it prevents excessive while loops in the home() function from menu.py.
2. convert()
    a. Converts the basic strategy’s move to be the same form as the player’s move, so they can be compared properly in the playGame() function.
3. playGame()
    a. Deals the starting hand to both the player and dealer. It waits for the player to choose a blackjack move which it then compares to see if it matches the one returned by the basic_strategy() function.

participant.py

This file defines the Player and Dealer classes. Specifically, the Player class is responsible for managing and interpreting all of the user's input while they play blackjack or the basic strategy minigame.

Functions:

1. allBust()
    a. Checks to see if every hand the user played this round has busted. This is required because the dealer only needs to add cards to their hand if they don't know the result of the round. If the user has busted, the dealer will win no matter what; therefore, they don't need to add any cards to their hand.
2. hit()
    a. Takes in a hand and deck object to retrieve a card from the deck using the deal() function and adds it to the hand by calling the addCard() function.
3. playHands()
    a. Calls the play() function for every hand the user has.
4. play()
    a. Handles all of the commands that can be inputted by the user and calls the function that satisfies the request made by the user.
5. split()
    a. Called if the user successfully chooses to split from the play() function. It splits the two cards stored in the Hand object into two separate hands.

visualization.py

The file provides a way for the system to visually represent the cards being played with.

Functions:

1. showCard()
    a. Takes in a card object and builds a visual representation of the card in the form of a string that is then printed out to the screen.
2. showHand()
    a. Takes in a hand object and prints out a visual representation of every card in the hand to the screen.