02-Milestone Project 2 - Walkthrough Steps Workbook

July 30, 2019

1 Milestone Project 2 - Walkthrough Steps Workbook

Below is a set of steps for you to follow to try to create the Blackjack Milestone Project game!

1.1 Game Play

To play a hand of Blackjack the following steps must be followed: 1. Create a deck of 52 cards 2. Shuffle the deck 3. Ask the Player for their bet 4. Make sure that the Player's bet does not exceed their available chips 5. Deal two cards to the Dealer and two cards to the Player 6. Show only one of the Dealer's cards, the other remains hidden 7. Show both of the Player's cards 8. Ask the Player if they wish to Hit, and take another card 9. If the Player's hand doesn't Bust (go over 21), ask if they'd like to Hit again. 10. If a Player Stands, play the Dealer's hand. The dealer will always Hit until the Dealer's value meets or exceeds 17 11. Determine the winner and adjust the Player's chips accordingly 12. Ask the Player if they'd like to play again

1.2 Playing Cards

A standard deck of playing cards has four suits (Hearts, Diamonds, Spades and Clubs) and thirteen ranks (2 through 10, then the face cards Jack, Queen, King and Ace) for a total of 52 cards per deck. Jacks, Queens and Kings all have a rank of 10. Aces have a rank of either 11 or 1 as needed to reach 21 without busting. As a starting point in your program, you may want to assign variables to store a list of suits, ranks, and then use a dictionary to map ranks to values.

1.3 The Game

1.3.1 Imports and Global Variables

** Step 1: Import the random module. This will be used to shuffle the deck prior to dealing. Then, declare variables to store suits, ranks and values. You can develop your own system, or copy ours below. Finally, declare a Boolean value to be used to control while loops. This is a common practice used to control the flow of the game.**

```
suits = ('Hearts', 'Diamonds', 'Spades', 'Clubs')
ranks = ('Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine', 'Ten', 'Jack', 'Queen
values = {'Two':2, 'Three':3, 'Four':4, 'Five':5, 'Six':6, 'Seven':7, 'Eight':8, 'Nine':9, 'Ten
          'Queen':10, 'King':10, 'Ace':11}
```

[ ]:
```
import random

suits = pass
ranks = pass
values = pass

playing = True
```

1.3.2 Class Definitions

Consider making a Card class where each Card object has a suit and a rank, then a Deck class to hold all 52 Card objects, and can be shuffled, and finally a Hand class that holds those Cards that have been dealt to each player from the Deck.

Step 2: Create a Card Class A Card object really only needs two attributes: suit and rank. You might add an attribute for "value" - we chose to handle value later when developing our Hand class.In addition to the Card's __init__ method, consider adding a __str__ method that, when asked to print a Card, returns a string in the form "Two of Hearts"

[ ]:
```
class Card:

    def __init__(self):
        pass

    def __str__(self):
        pass
```

Step 3: Create a Deck Class Here we might store 52 card objects in a list that can later be shuffled. First, though, we need to *instantiate* all 52 unique card objects and add them to our list. So long as the Card class definition appears in our code, we can build Card objects inside our Deck __init__ method. Consider iterating over sequences of suits and ranks to build out each card. This might appear inside a Deck class __init__ method:

```
for suit in suits:
    for rank in ranks:
```

In addition to an __init__ method we'll want to add methods to shuffle our deck, and to deal out cards during gameplay. OPTIONAL: We may never need to print the contents of the deck during gameplay, but having the ability to see the cards inside it may help troubleshoot any problems that occur during development. With this in mind, consider adding a __str__ method to the class definition.

[ ]:
```
class Deck:

    def __init__(self):
        self.deck = []  # start with an empty list
        for suit in suits:
            for rank in ranks:
                pass

    def __str__(self):
```

```
        pass

    def shuffle(self):
        random.shuffle(self.deck)

    def deal(self):
        pass
```

TESTING: Just to see that everything works so far, let's see what our Deck looks like!

[ ]:
```
test_deck = Deck()
print(test_deck)
```

Great! Now let's move on to our Hand class.

Step 4: Create a Hand Class In addition to holding Card objects dealt from the Deck, the Hand class may be used to calculate the value of those cards using the values dictionary defined above. It may also need to adjust for the value of Aces when appropriate.

[ ]:
```
class Hand:
    def __init__(self):
        self.cards = []  # start with an empty list as we did in the Deck class
        self.value = 0   # start with zero value
        self.aces = 0    # add an attribute to keep track of aces

    def add_card(self,card):
        pass

    def adjust_for_ace(self):
        pass
```

Step 5: Create a Chips Class In addition to decks of cards and hands, we need to keep track of a Player's starting chips, bets, and ongoing winnings. This could be done using global variables, but in the spirit of object oriented programming, let's make a Chips class instead!

[ ]:
```
class Chips:

    def __init__(self):
        self.total = 100  # This can be set to a default value or supplied by a␣
 ↪user input
        self.bet = 0

    def win_bet(self):
        pass

    def lose_bet(self):
        pass
```

1.3.3 Function Defintions

A lot of steps are going to be repetitive. That's where functions come in! The following steps are guidelines - add or remove functions as needed in your own program.

Step 6: Write a function for taking bets Since we're asking the user for an integer value, this would be a good place to use try/except. Remember to check that a Player's bet can be covered by their available chips.

[ ]:
```
def take_bet():

    pass
```

Step 7: Write a function for taking hits Either player can take hits until they bust. This function will be called during gameplay anytime a Player requests a hit, or a Dealer's hand is less than 17. It should take in Deck and Hand objects as arguments, and deal one card off the deck and add it to the Hand. You may want it to check for aces in the event that a player's hand exceeds 21.

[ ]:
```
def hit(deck,hand):

    pass
```

Step 8: Write a function prompting the Player to Hit or Stand This function should accept the deck and the player's hand as arguments, and assign playing as a global variable. If the Player Hits, employ the hit() function above. If the Player Stands, set the playing variable to False - this will control the behavior of a while loop later on in our code.

[ ]:
```
def hit_or_stand(deck,hand):
    global playing  # to control an upcoming while loop

    pass
```

Step 9: Write functions to display cards When the game starts, and after each time Player takes a card, the dealer's first card is hidden and all of Player's cards are visible. At the end of the hand all cards are shown, and you may want to show each hand's total value. Write a function for each of these scenarios.

[ ]:
```
def show_some(player,dealer):

    pass

def show_all(player,dealer):

    pass
```

Step 10: Write functions to handle end of game scenarios Remember to pass player's hand, dealer's hand and chips as needed.

[ ]:
```
def player_busts():
    pass

def player_wins():
    pass

def dealer_busts():
    pass

def dealer_wins():
```

```
    pass

def push():
    pass
```

1.3.4 And now on to the game!!

[ ]:
```
while True:
    # Print an opening statement


    # Create & shuffle the deck, deal two cards to each player



    # Set up the Player's chips


    # Prompt the Player for their bet


    # Show cards (but keep one dealer card hidden)


    while playing:  # recall this variable from our hit_or_stand function

        # Prompt for Player to Hit or Stand


        # Show cards (but keep one dealer card hidden)


        # If player's hand exceeds 21, run player_busts() and break out of loop


            break

    # If Player hasn't busted, play Dealer's hand until Dealer reaches 17


        # Show all cards

        # Run different winning scenarios


    # Inform Player of their chips total
```

```
    # Ask to play again

        break
```

And that's it! Remember, these steps may differ significantly from your own solution. That's OK! Keep working on different sections of your program until you get the desired results. It takes a lot of time and patience! As always, feel free to post questions and comments to the QA Forums. # Good job!