OOP Design II

Instructor: Paruj Ratanaworabhan

Let’s Play Blackjack

games.washingtonpost.com/games/blackjack

Goal and How to Meet It

- Beat the dealer
- How do you beat the dealer?
    - By drawing a hand value that is higher than the dealer’s hand value
    - By the dealer drawing a hand value that goes over 21 and your drawing does not
    - By drawing a hand value of 21 on your first two cards, when the dealer does not
- How do you lose to the dealer?
    - Your hand value exceeds 21 and the dealer hand value does not
    - The dealers hand has a greater value than yours at the end of the round
- Some restrictions:
    - **Hand value cannot be below 16; if so, must draw more cards**

Blackjack

CARD VALUES




= 10

= 1 OR 11


2


3

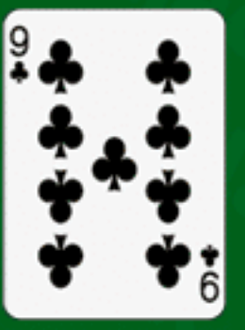

4


5


6


7


8


9

Programming Blackjack: Overview

- Two players, you and the computer
- Three outcomes, win, tie, or loose
- Initially, you and the computer are dealt two cards. You get to see only one card of the computer's hand whereas your hand exposes all the cards
- You have to make a decision to stay or take more cards
- Once you are done, the computer proceeds to stay or get more cards

OOP Thinking

- How should we represent a game of Blackjack?
- The game uses a deck of cards so having a class that a card deck object can be created should be handy
- There should also be another class that represents the game of Blackjack itself
    - Describing the state of this class will involve the objects of the card type

Programming Blackjack: Details

- How to represent a deck of 52 cards?
- How to shuffle a deck of 52 cards?
- How to draw initial hands for a player and the computer?
- How to calculate the value of a hand?
- How to draw more cards and recalculate the hand value?
- How the computer makes a decision to draw more cards or to stay?
- How to make a decision about the final outcome?
- How to print out cards?

Representing a Deck of Cards

Representing a deck of cards with a list of strings

```
class CardDeck:
    '''This class describes a deck of 52 cards with 4 suits and 13 ranks
    '''
    def __init__(self):
        SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
        RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack',
'Queen', 'King', 'Ace']
        deck = []
        for rank in RANKS:
            for suit in SUITS:
                card = rank + ' ' + suit
                deck += [card]
        self.card_deck = deck

    # methods for performing operations on a deck of cards to follow
```

Operations on a Deck of Cards

```
class CardDeck:
    '''This class describes a deck of 52 cards with 4 suits and 13 ranks
    '''

    # def __init__(self): code

    # list of methods
    # must be able to shuffle it
def shuffle(self):
    import random
    n = len(self.card_deck)
    for i in range(n):
        r = random.randrange(i, n)
        temp = self.card_deck[r]
        self.card_deck[r] = self.card_deck[i]
        self.card_deck[i] = temp
```

Operations on a Deck of Cards

```
class CardDeck:
    '''This class describes a deck of 52 cards with 4 suits and 13 ranks
    '''

    # def __init__(self): code

    # list of methods
    # must be able to shuffle it
    # def shuffle(self):

    # must be able to draw n cards from it
    # def draw_cards(n):
```

Representing Blackjack

State of a Blackjack game is described by:

- player hand as a list of cards
- computer hand as a list of cards
- player possible hand values as list of integers
- computer possible hand values as list of integers
- player hand status as [stay or dead or can_stay or must_draw_more]
- computer hand status as [stay or dead or can_stay or must_draw_more]

```
class Blackjack:
    '''This class describes the game of Blackjack itself
    '''
    def __init__(self):
        deck = CardDeck()
        deck.shuffle()
        self.bj_deck = deck
        self.player_hand = []
        self.computer_hand = []
        self.player_hand_value = … # can have multiple values when hand involves Aces
        self.computer_hand_value = ... # can have multiple values when hand involves Aces
        self.player_hand_status =  # [stay or dead or can_stay or must_draw_more]
        self.computer_hand_status = # [stay or dead or can_stay or must_draw_more]
```

Operations for Blackjack

```
class Blackjack:
    '''This class describes the game of Blackjack itself
    '''
    # def __init__(self):

    def start(self):
        '''This method starts the game by drawing from the card deck (represented by
self.bj_deck) two cards for player and two cards for computer; then it proceed to update all
the values and statuses for both hands
        '''

    def adjust_player_hand():
        '''This method draws an additional card and reevalute the value and status of the
player hand
        '''

    def adjust_computer_hand():
        '''This method draws an additional card and reevalute the value and status of the
computer hand
        '''

    def display_player_hand():
        '''This method reveals the player hand
        '''

    def display_computer_hand():
        '''This method reveals the computer hand
        '''

    def decision():
        '''This method makes a decision if the player wins, looses, or ties with the
computer
        '''
```

Playing Blackjack

```
new_BJ = Blackjack()
new_BJ.start()

# player plays
while new_BJ.player_hand_status != 'can_stay':
    new_BJ.adjust_player_hand()
    new_BJ.display_player_hand()

while new_BJ.player_hand_status == 'can_stay':
    to_stay_or_not = input("stay or not")
    if to_stay_or_not == False:
        new_BJ.adjust_player_hand()
        new_BJ.display_player_hand()
    else:
        new_BJ.player_hand_status = ‘stay'

# computer plays
while new_BJ.computer_hand_status != 'can_stay':
    new_BJ.adjust_computer_hand()
    new_BJ.display_computer_hand()
while new_BJ.computer_hand_value < new_BJ.player_hand_value and new_BJ.computer_hand_status
== 'can_stay':
    new_BJ.adjust_computer_hand()
    new_BJ.display_computer_hand()

new_BJ.decision()
```

How to calculate the value of a hand?

- Must consider:
  - Extract the RANKS part of the string drawn from deck list and calculate the value accordingly
  - Remember: ACE could be one or eleven, so this case must be carefully dealt with

Blackjack

CARD VALUES




= 10

= 1 OR 11


2


3

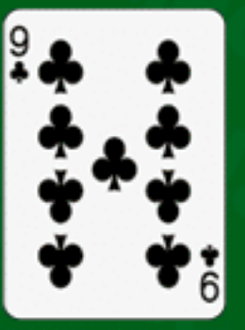

4


5


6


7


8


9

How the computer makes a decision to draw more cards or to stay?

- If the computer hand value is less, it must fight!

- If the player hand is “dead”, it just needs to be at or go over the threshold value

How to make a decision about the final outcome?

- Comparing the values of computer and player hands
- Hand value for each must be in one of these two states:
  - Above 21 or
  - Greater than or equal the threshold and less than or equal 21

How to print out cards in a hand?

- See the following example:

```
>>> SUITS_sym = ["\u2663", "\u2666", "\u2660", "\u2665"]

>>> print(SUITS_sym)

['♣', '♦', '♠', '♥']
```

A Sample Session

```
Let's play Blackjack!
Your hand: 4♣ 8♦
Computer hand J♥
Your hand: 4♣ 8♦ 2♠
Your hand: 4♣ 8♦ 2♠ 6♦
More cards? No
Computer hand: J♥ 8♥ 9♦
You win!
Play a new round: Yes

Let's play Blackjack!
Your hand: 10♣ 9♦
Computer hand 8♥
More cards? No
Computer hand: 8♥ 3♥ Q♦
You loose!
Play a new round: Yes

Let's play Blackjack!
Your hand: 10♥ 5♦
Computer hand A♦
Your hand: 10♥ 5♦ 4♥
More cards? No
Computer hand A♦ 7♦ A♣
You tie with the computer!
Play a new round: No
```

In Summary

- We learn about designing a Blackjack OO program
- Use two classes, CardDeck and Blackjack
- BlackJack utilizes the CardDeck object in its gaming operations