Blackjack Project

Algorithms and python programming

Students: Abdelkoddous Hounkara, Vasileios Skarleas

Introduction:
On the UE course of INF101 under the general idea of learning algorithms and programming in Python, we are forced to develop a blackjack game by using Python and following the different programming techniques as well as skills that we developed during this semester. Below are written the indications that we needed to follow, the procedure that we followed in order to resolve any problems, as well as a summary of the problems / difficulties that we met. Our goal through this project is to make it even more clear how to connect many different functions in a Python programming environment while we are staying focused on the indications given by the project's paper.

Status:

- Total hours of development: 32 hours
- Total lines of codes: 744

Indications:
We need to create a blackjack game by using the Python programming language. The project is divided into 4 different sections, where, by combining them we have a completed result. More specifically:

A. Initialisations
B. Game Management
C. Some intelligence
D. UI (graphical interface)

We note that parts A and B are directly connected the one to other, while C provides more functionality onto the part B, and finally the part D is a complete separate task from all the others in terms of functionality.

As blackjack is a cards game, the creation of deck or decks of cards is essential, as well as the possibility to count the different points gained from every player according to the following criteria:

- Cards numbered from 2 to 10 are valued as many points as their number (so between 2 and 10 points)
- The ace is valued either 1 or 11 points, at the player's choice
- Figures (jack, queen, king) are valued 10 points each

The data structure provides two different options so that we can play several games in a row. This includes the management of many different players' scores. This is why it's preferred to use lists or dictionaries for saving and retrieving data.

- Option 1: scores, number of victories, and money gains of each player are stored in lists, so they can be accessed from the player number (index in the corresponding list).

- Option 2: scores, number of victories, and money gains of players are stored in dictionaries whose keys are the players' names, so they can be accessed with the player's name instead of the player's number.

Procedure:
DISCLAIMER: We followed the steps as mentioned onto the project document and indications but maybe there are sections that we have changed the data structure to make the final game more efficient, especially on our version 2 of code.

As team, we decided to work to the corresponding workflow below:

- Organization, transparency and communication are our standards for a good team work.
- According to the mantra mentioned above, we used the softwares mentioned below:
    1. GitHub: Includes our private repository with all the code that has been submitted by the team members. It tracks the live versions that every team member is watching every time on his personal desktop workstation. Also, GitHub provides a great way to visualize any changes and to restore previous versions of the code.
    2. VSCode: Instead of the Python idle, we decided to load python and work with this programming language, in our TD, TP, Caseine and current project, on VSCode since it is a universal development app, with some great extensions to integrate onto the workflow and to visualize better the different commands. In addition to that, it offers a direct connection to our GitHub repository.
    3. PythonTutor: Online program service indicated by the UE that helps to the deeper understanding of the corresponding code that we are examining every single time.
    4. W3Schools: Used for classes documentation that we integrated for part c on our separate version (explanation at paragraph Difficulties/Project – section 11).
- From the first view of the project, we knew that the part D would be the most difficult one and there was a possibility that we couldn't be able to make it so far. So, our goal was set to complete as better as possible the rest of the tasks.
- For every step on the different four sections mentioned above, we decided to follow the "exams technique". This means, that we will follow the exact steps by including the required elements mentioned on the specific task (function, returned value, required arguments, lists, etc.) as part of project's understanding. The rest of the code will get completed according to the personal point of view through the understanding process of the task. This will allow us to have the biggest understanding that is possible, for this demanding project.
- We decided to set as default value for Ace the number 11 and instead of posting the question inside the valueCard module, we are asking the player at the moment of his turn to decide before we show up or not the next card.
- Via object-oriented programming techniques, we defined the different functions in a way that enables the automatic update and synchronization of lists and dictionaries,

without the need to initialize any variables. Everything is automatically handled by the players list that is created by the system. (referred to version 2 of our program)

Difficulties/Problems:

In this section we are presenting the different challenges that we met and the workflows around them in order to be solved:

1. The biggest difficulty that we had was the concept the idea of the game. Since we had never played that game, nor having experience with these types of games, it was extremely difficult to understand what we are expecting from the program to do, something that corresponds at how will combine all the functions that we initialized on part A and result at least at one game play on part B. On the other hand, the UNO game for instance that we have in our copies, was a game that we know how it works and what changes to do in the program in order to make it work correctly.
2. One of the first challenges was the introduction and use of dictionaries. The moment that we started the development of this project, we didn't have done it in our course. Thanks to Python's documentation, the different internal functions() of dictionaries in pythons and key words like "keys" and "values" were used as expected on the different sections (like initPlayers()).
3. Another challenge was the initialization of cards' deck. According to different resources, to most direct way is to use classes in Python which will then also help to the UI (fourth part of the project) by calling the executing the specified task. But we wanted a more direct way that is closed to our recent knowledges in python, and that's why in our workflow we used lists to initialize the deck.
4. It was challenging to write the drawPick() function as long as we don't know how a real blackjack game is played. This is also connected with the gameTurn() and completeGame() functions. YouTube was a very helpful tools in this understanding and searching procedure but again, without the real-life experience, we think we did not succeed.
5. Onto the firstTurn() function, even if we have constructed the drawCard() function to send the requested amount of cards in x variable from the p list of cards, when we were compacting these two cards that are requested on firstTurn() function onto a new list and then we were performing a "for" loop to take the different values, when we're removing every specific card that was already triggered from deck, we were receiving an error that was changing the length of the two cards list that we were creating in very repeation of this section of the code. This is why we divided the process in one task that repeats in total 2 times for every player.
6. The valueCard() function sometimes was causing the firstTurn() function to crush due to NoType defined error in the returned values. In order to resolve the issue, we changed the examination formula of specific cards value in a generic one which permits not to worry for any typos on the defined lists that we create for the decks.
7. For playerTurn() function is was needed to parse the modified deck after the first turn that we count the initial points and this is why we are returning two elements onto the firstTurn() function, the dictionary with the players and the scores, as well as the modified deck of cards.
8. It could be possible to use GUI to create a real graphical interface for the section D of this project. Since we didn't cover it in our lessons, we didn't have the time to learn

in depth the GUI functions. Instead, we created a console-based UI which prints out the cards and visualizes them which is called cardVisualization() – more below.

9. Since the initial deck of cards was a list of letters and numbers in one string corresponding to one card, in every step that there is modification of the deck, we are synchronizing it with a list that includes keyboards symbols (ASCII), which allows to use this synchronized list in order to print out the cards onto the console.
10. In order to count the victories we know that we need to use the winner() function but we are not forced to do it. We the different data structure that we created; in the end of every round, we receive the dictionary with the remaining players. With the use of winner() we determine the winner between the remaining players instead and we parse these data in a local list that is added via the main program. When the players want in general to terminate the game (main program), then we filter this local list and we take the final winner instead.
11. In order to minimize our first version of the code, the betting functionality was added directly to our version 2 of the program that also includes, the betting and the personalization functionality.
12. On the first version of the code, during debugging process, we set the program to run in total 3 successive rounds at completeGame() function, since it was very time consuming to running the whole process until we reach again at the section that we had a bug to resolve. But, very easily we can transform it to an infinite loop until only remains one player.
13. The continues() function on version 1 of the code, just asks if the players want to continue the game. On the version 2 of the code we have created an advanced feature to remove players from the Players dictionary that they want to withdraw. We have similar requests on the two versions, but we don't have the same actions.
14. For the section C of the project, we knew that had to initialize the croupier as an existing player on the game. We decided though to create a whole new data structure and a program based on object-oriented programming as mentioned in our introduction. The program includes the required intelligence for the croupier, the personalization of the game, as well as the betting process. Access the explanation of our version 2 and 3 of the code at https://dev.madebyvasilis.site/blackjack-2021 .
    a. By letting our imagination free, we thought that we could modify the program to become even more fun by adding a split_deck() function that will count how many cards has the player and ask him if he wants to use his two hands to hold the virtual cards ☺. Available functionality on version 3 of the program.
    b. Another possible scenario could be to add the possibility for players to use an insurance while they are playing. Insurance() is called on version 3 of the program.

Project's code:

We are attaching our project's code divided into the different sections of the project mentioned on this document's introduction. For the section D, we have integrated a partially autonomous UI printing system inside the corresponding functions where drawCard() is used, as explained on points 8 and 9 above.

Version 1:

```
import random
import sys
import copy

#This creates the deck
def deck():
    values = ['2','3','4','5','6','7','8','9','10','Jack','Queen','King','Ace']
    suites = ['Hearts', 'Clubs', 'Diamonds', 'Spades']
    deck = [[v + ' of ' + s] for s in suites for v in values]
    return deck

#This give value to the cards
def valueCard(card):
    if "2" in card:
        return 2
    elif "3" in card:
        return 3
    elif "4" in card:
        return 4
    elif "5" in card:
        return 5
    elif "10" in card:
        return 10
    elif "6" in card:
        return 6
    elif "7" in card:
        return 7
    elif "8" in card:
        return 8
    elif "9" in card:
        return 9
    elif "Q" in card or "K" in card or "J" in card:
        return 10
    elif "A" in card: #User is asked in the UI to select the final value for Ace
        return 11

#This creates the overal final deck that will be used to the whole game
def initStack(n):
    finalDeck = []
    value = deck()
```

```
    for i in range(n):
        for k in range(len(value)):
            finalDeck.append(value[k])
    random.shuffle(finalDeck)
    return finalDeck

#This takes the coresponding number of cards from the predefined final deck (p here) and return this list of card/s
def drawCard(p,x):
    if len(p) == 1 or x==':
        return p
    elif len(p) < x:
        return p
    elif x == 1:
        var =  p[0]
        p = list(var)
        return p
    else:
        for i in range((len(p)-(x-1))-1):
            p.pop(x)
        return p

#This is used to initilize the players names and personalize the game
def initPlayers(n):
    players = []
    id = 1
    for id in range(n):
        while True:
            name = str(input('Name of player ' + str(id+1) + ': '))
            if name != ':
                players.append(name)
                print("Nice to meet you ", name,"!", sep="")
                break
            else:
                print('Please, tell me your name.')
    print("We are ready. Let's start!\n")
    return players

#This creates the dictionary that we are using in all the functions below
def initScores(players,v=0):
    if v != 0:
```

```
        dictionary = dict.fromkeys(players, v)
    else:
        dictionary = dict.fromkeys(players, 0)
    return dictionary #usually mentioned scores on the other functions below


def cardVisualization(card):
    cardOutput = card[0]
    elem = []
    if "Hearts" in card:
        elem.append("❤")
    elif "Diamonds" in card:
        elem.append("♦")
    elif "Spades" in card:
        elem.append("♠")
    else:
        elem.append("♣")
    if "2" in cardOutput:
        elem.append("2")
    elif "3" in cardOutput:
        elem.append("3")
    elif "4" in cardOutput:
        elem.append("4")
    elif "5" in cardOutput:
        elem.append("5")
    elif "6" in cardOutput:
        elem.append("6")
    elif "7" in cardOutput:
        elem.append("7")
    elif "8" in cardOutput:
        elem.append("8")
    elif "9" in cardOutput:
        elem.append("9")
    elif "10" in cardOutput:
        elem.append("10")
    elif "Jack" in cardOutput:
        elem.append("J")
    elif "Queen" in cardOutput:
        elem.append("Q")
    elif "Ace" in cardOutput:
        elem.append("A")
```

```
    elif "King" in cardOutput:
        elem.append("K")
    return elem

#This creates the first view of players points
def firstTurn(players):
    scores = initScores(players, v=0)
    n = len(players)
    deck = initStack(n)
    for h in range(len(players)):
        res = 0
        #print("For", players[h], "we have:")
        for repeat in range(2):
            card = drawCard(deck, 1)
            #identifiers = cardVisualization(card)
            #print(' ┌─────────┐ ')
            #print(f'| {identifiers[0]}     |')
            #print('|         |')
            #print(f'|   {identifiers[1]}   |')
            #print('|         |')
            #print(f'|    {identifiers[0]}  |')
            #print(' └─────────┘ ')
            deck.remove(card)
            if valueCard(card[0]) == 1:
                print("Your card is ",card[0], "and you need to select its value")
                choice = int(input("11 or 1 ? "))
                if choice == 1:
                    res = res + 1
                else:
                    res = res + 11
            else:
                res = res + valueCard(card[0])
        scores[players[h]] = res
    return scores, deck

#This determines who is the winner based on every game round
def winner(scores):
    score = list(scores.values())
    people = list(scores.keys())
    max_score = 0
```

```
    dictionary = {}
    for i in range(len(score)):
        if score[i] <= 21 and score[i]>max_score:
            max_score = score[i]
            id = i
    max_people = people[id]
    for i in range(len(people)):
        d1 = {str(people[i]):0}
        dictionary.update(d1)
    return max_people, dictionary


def continues(): #It is like withdraw
    default = True
    print("Do you want to continue the game ?")
    choice = str(input("yes/no : "))
    while default == True:
        if choice == "no":
            default = False
            return False
        elif choice == "yes":
            default = False
            return True
        else:
            print("I dont understand your input. Please type if you wamnt to continue teh game")
            choice = str(input("yes/no : "))


def playerTurn(j, players):
    data = firstTurn(players) #data in position 0 is the dictionary with the scores. In position 1 we have the modified
deck
    points = data[0][j]
    if points<21:
        print("It's", j, "turn. You have", points, "points.")
        choice = str(input("hit/stand?"))
        while True:
            if choice == "stand":
                return data[0], data[1]
                break
            elif choice == "hit": #Player continues
                print("OK, let's continue.")
```

```
            card = drawCard(data[1], 1)
            data[1].remove(card)
            print("Here is your next card:")
            value = valueCard(card[0])
            identifiers = cardVisualization(card)
            print(' ┌─────────┐ ')
            print(f'| {identifiers[0]}     |')
            print('|       |')
            print(f'|   {identifiers[1]}   |')
            print('|       |')
            print(f'|    {identifiers[0]}  |')
            print(' └─────────┘ ')
            res = points + value
            if res > 21:
                print("Unfortunately,", j,"lost since his/her points were", res)
                data[0].pop(j)
                return data[0], data[1]
                break
            else:
                data[0][j] = res
                return data[0], data[1]
                break
        else:
            print("I don't understand.")
            choice = str(input("hit/stand? "))
    else:
        print("Unfortunately,", j,"lost since his/her points were", points)
        data[0].pop(j)
        return data[0], data[1] #1 is deck, 0 is players dictionary with scores.

def turn(j, players, deck):
    if players[j]<21:
        print("It's", j, "turn. You have", players[j], "points.")
        choice = str(input("hit/stand? "))
        while True:
            if choice == "stand":
                return players, deck
                break
            elif choice == "hit": #Player continues
                print("OK!")
```

```
            card = drawCard(deck, 1)
            deck.remove(card)
            print("Here is your next card:")
            value = valueCard(card[0])
            identifiers = cardVisualization(card)
            print(' ┌─────────┐ ')
            print(f'| {identifiers[0]}      |')
            print('|         |')
            print(f'|   {identifiers[1]}   |')
            print('|         |')
            print(f'|    {identifiers[0]}  |')
            print(' └─────────┘ ')
            res = players[j] + value
            if res > 21:
                print("Unfortunately,", j,"lost since his/her points were", res)
                players.pop(j)
                return players, deck
                break
            else:
                players[j] = res
                return players, deck
                break
        else:
            print("I don't understand.")
            choice = str(input("hit/stand? "))
    else:
        print("Unfortunately,", j,"lost since his/her points were", players[j])
        data[0].pop(j)
        return data[0], data[1]

def gameTurn(data, players): #Receives the dictionary of players every time
    for i in range(len(list(data.keys()))):
        lamda = list(data.keys())
        j = lamda[i]
        save = playerTurn(j, players)
        players = list(save[0].keys())
    return save


def gameOver(data): #Receives the dictionary of players every time
```

```
    number = len(list(data.keys())) #SOS - There is a problem here
    if number >= 1: #One single player on the dictionary remains
        return True
    else:
        return False

def completeGame(data, players): #Receives the dictionary of players every time
    default = False
    max = 0
    id = 2
    ids = 1
    print("Round no 1")
    play = gameTurn(data, players)
    over = gameOver(play[0])
    if over == False:
        print("Game over")
        print("No remaining players on the game.")
    elif over == True:
        continueing = continues()
        if continueing == True:
            win = winner(play[0])
            win[1][win[0]]= win[1][win[0]]+1
            updatedDeck = play[1]
            players = play[0]
            if len(list(players.keys())) > 1:
                while id > 0:
                    print("Round no", ids + 1)
                    playing = copy.deepcopy(players)
                    for j, value in players.items():
                        newRound = turn(j, playing, updatedDeck)
                    over = gameOver(newRound[0])
                    if over == False:
                        print("Game over")
                        print("No remaining players on the game.")
                        sys.exit()
                    elif over == True:
                        win = winner(newRound[0])
                        win[1][win[0]]= win[1][win[0]]+1
                        updatedDeck = newRound[1]
                        players = newRound[0]
```

```
                    playing = players
                    id = id -1
                continueing = continues()
                if continueing == False:
                    print("We have played", ids,"rounds in total")
                    for g in range(len(list(win[1].keys()))):
                        if max < list(win[1].values())[g]:
                            max = list(win[1].values())[g]
                            person = list(win[1].keys())[g]
                    print("From the remaining", len(list(newRound[0].keys())), "players, the winner is",person, "with",
max, "total victories.")
                    print("\nThanks for playing our BlackJack game. See yoou soon, bye bye!")
                    sys.exit()
            print("We have played three rounds in total") #Of course we can write so that it repears until we have
one player but this approach helped us hrough debugging proces
            for g in range(len(list(win[1].keys()))):
                if max < list(win[1].values())[g]:
                    max = list(win[1].values())[g]
                    person = list(win[1].keys())[g]
            print("From the remaining", len(list(newRound[0].keys())), "players, the winner is",person, "with", max,
"total victories.")
        else:
            print("We have a winner!")
            if max < list(win[1].values())[0]:
                max = list(win[1].values())[0]
            print("The winner is", list(win[1].keys())[0], "with total victories", max)
    else:
        print("Alright, the game is terminated. Bye bye!")


#Main Program
n = int(input("How many players ? "))
print("Let's personalize your game a little bit.") #Via object programming techniques, all the inializations are done
automaticly
players = initPlayers(n)
data = initScores(players,v=0)
completeGame(data, players)
```

Version 2:

```
import random
import sys
```

```
Deck = {'Ace': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
        'Jack': 10, 'Queen': 10, 'King': 10}
Suit = ['of Clubs', 'of Spades', 'of Hearts', 'of Diamonds']
Croupier = {'Name': 'MAX', 'Deck1': [], 'c_values1': [], 'Play': True, 'Ace': False, 'BJ': False, 'Score1': 0}
Template = {'Name': '', 'Money': 0, 'Bet': 0, 'Active': True, 'Play': True, 'BJ': False, 'Double': False, 'Ace': False,
            'Score1': 0, 'insurance': False, 'half_bet': 0}
Players = {}
Config = {'Number of players': 0, 'Money': 100, 'Minimum_bet': 5, 'show': True, 'game': 0, 'deck_size': 0,
          'max_deck': 0}
used_cards = {}
Croupier_names = ['Max', 'Ilias', 'Joey', 'Jenk', 'Asher', 'Sophie', 'Johnny', 'Iris', 'Kennedy', 'Nike']

def startGame():
    Croupier['Name'] = random.choice(Croupier_names)
    print("Hello, my name is {}. I'll be your Croupier for this session.".format(Croupier['Name']))
    print('The prizes are:')
    print('- Blackjack pays 3 to 2.')
    print('- Wins pays 1 to 1.')
    print('- Insurance pays 2 to 1.')
    configuration()

def configuration():
    print('\nLets personalize your game a little bit')
    print('Do you prefer default rules, or would you rather have a custom set of rules?')
    while True:
        print('The default rules are: Minimum bet = {}¢; Funds available to each player = {}¢; Croupier show one
card.'.format(
            Config['Minimum_bet'], Config['Money']))
        configurationuration = input("default/custom: ").lower() #In order to prevent any error, the lower() predefines
for what we need to filter bellow
        if configurationuration not in ['default', 'custom']:
            print('Please, enter a valid input.')
        elif configurationuration == 'default':
            numberPlayers()
        else:
            while True:
                min_bet = input("How much should minimum bet be? ")
                if min_bet.isdigit() and 0 < int(min_bet) < 100: #We are not forced to have the <100 option here
                    Config['Minimum_bet'] = int(min_bet)
```

```
                break
        while True:
            funds = input("How much funds should each player have? ")
            if funds.isdigit() and Config['Minimum_bet'] * 2 < int(funds): #Minimu_bet just changed above
                Config['Money'] = int(funds)
                break
        while True:
            print('Croupier should show you one card?')
            show = input("yes/no: ").lower()
            if show == 'yes':
                Config['show'] = True
                break
            if show == 'no':
                Config['show'] = False
                break
        numberPlayers()

def numberPlayers():
    n = input("How many players ? ")
    if n.isdigit() and 1 <= int(n):
        Config['Number of players'] = int(n)
        Config['max_deck'] = Config['Number of players']

        for x in range(1, Config['Number of players'] + 1):
            new_player = 'Player ' + str(x) #This will get personalized via the initPlayers() function
            template_new = dict(Template)
            Players.setdefault(new_player, template_new) #FRom python docs: The setdefault() method returns the value of the item with the specified key. If the key does not exist, insert the key, with the specified value
            Players[new_player].setdefault('Deck1', [])
            Players[new_player].setdefault('c_values1', [])
        initPlayers()
    else:
        print('Enter a valid input.')
        numberPlayers()

def initPlayers():
    for x in Players: #The players dictionary includes a dictionary created from the template dectionary as initiliazed from the numberPlayers function
        if Players[x]['Active']: #Safety valve that verifies that the players dictionary has elements or where it has to stop asking since the dictionary elemnts do not have IDs like the lists
```

```
        while True:
            Players[x]['Name'] = input("Name of player " + list(x)[7] + ": ") #I can explain why I have added this seven here in this corresponding list
            if Players[x]['Name'] != '':
                print('Nice to meet you {}!'.format(Players[x]['Name']))
                Players[x]['Money'] = Config['Money']
                break
            else:
                print('Please, tell me your name.')
    print("We are ready. Let's start!")
    betting()

def betting():
    Config['game'] = Config['game'] + 1 #The
    print('\nGame number {}'.format(Config['game']))
    if Config['game'] % 10 == 0: #When the croupier has lost specific amount of times, we are channging croupier
        while True:
            new_croupier = random.choice(Croupier_names)
            if new_croupier != Croupier['Name']:
                Croupier['Name'] = new_croupier
                print('My turn is over. I introduce you to your new croupier, {}'.format(Croupier['Name']))
                print('Have fun!\n')
                break
    print('\nTime to give your bets')
    for x in Players:
        if Players[x]['Active']:
            while True:
                print('\nPlease, {}. Place a bet! You can go up to {}'.format(Players[x]['Name'], Players[x]['Money']))
                bet = input("Your bet: ")
                if bet.isdigit() and Config['Minimum_bet'] <= int(bet) <= 500 and int(bet) <= Players[x]['Money']:
                    Players[x]['Bet'] = int(bet)
                    print("Your bet has been registered.")
                    break
    firstTurn()

def firstTurn():
    for x in Players:
        if Players[x]['Active']:
            drawCard(2, Players[x], 'Deck1', 'c_values1')
    drawCard(2, Croupier, 'Deck1', 'c_values1')
```

```
    for x in Players:
        if Players[x]['Active']:
            print(Players[x]['Name'] + ' these are your cards:')
            print_deck(Players[x], 'Deck1')
    if Config['show']:
        print("The croupier, {}, has {} and one hidden card.".format(Croupier['Name'], Croupier['Deck1'][0]))
    else:
        print("The croupier, {}, has two hidden cards.".format(Croupier['Name']))
    for x in Players:
        if Players[x]['Active'] and Players[x]['Play']:  # Check for Blackjack first on both sides
            player = Players[x]
            cards = player['c_values1']
            if (cards[0] == 'Ace' and Deck.get(cards[1]) == 10) or (Deck.get(cards[0]) == 10 and cards[1] == 'Ace'):
                player['BJ'] = True
                print("Congratulations {}! You have Blackjack!".format(player['Name']))
                player['Play'] = False
            cards = Croupier['c_values1']
            if (cards[0] == 'Ace' and Deck.get(cards[1]) == 10) or (Deck.get(cards[0]) == 10 and cards[1] == 'Ace'):
                Croupier['BJ'] = True
                print('I have Blackjack!')
                print_deck(Croupier, 'Deck1')
                winner()  # if Croupier has blackjack, no need to look more
    for n in Players:
        player = Players[n]
        hit_stand(Players[n], 'Deck1', 'c_values1', 'Score1', 'Play')
        if Players[n].get('Double'):
            hit_stand(Players[n], 'Deck2', 'c_values2', 'Score2', 'Double')
    croupier()


def croupier():
    print("\nIt's my turn.")
    while Croupier['Play']:
        val = valueCardSum(Croupier['c_values1'])
        for card_v in Croupier['c_values1']:
            if card_v == 'Ace':
                Croupier['Ace'] = True
        print('My cards are:')
        print_deck(Croupier, 'Deck1')
        if Croupier['Ace'] and (val + 10) <= 21:
            print('Hard value: {}'.format(val))
```

```
            print('Soft value: {}\n'.format((val + 10)))
        else:
            print("Its values is: {}\n".format(val))
        if val > 21:
            print('Bust! My hand is over 21.')
            Croupier['Play'] = False
            Croupier['Score1'] = val
        elif Croupier['Ace'] and 17 <= (val + 10) <= 21:
            print('I stand.')
            Croupier['Score1'] = val + 10
            Croupier['Play'] = False
        elif 17 <= val <= 21:
            Croupier['Score1'] = val
            print('I stand.')
            Croupier['Play'] = False
        elif Croupier['Ace'] and (val + 10) < 17:
            print('I hit for another card.')
            drawCard(1, Croupier, 'Deck1', 'c_values1')
        elif val < 17:
            print('I hit for another card.')
            drawCard(1, Croupier, 'Deck1', 'c_values1')
    winner()

def winner():
    for x in Players:
        if Players[x]['Active']:
            player = Players[x]
            if Croupier['BJ']:
                if player['BJ']:
                    print('{}. You recover your bet of {}¢.'.format(player['Name'], player['Bet']))
                if player['insurance'] and 0 < player.get('half_bet', 0):
                    print("{} your insurance covers your bet and you win {}¢".format(player['Name'],
                                                                                     player['half_bet'] * 2))
                    player['Money'] += player['half_bet'] * 2
                else:
                    player['Money'] -= player['Bet']
                    print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
            else:  # Croupier['BJ'] is False
                if player['BJ']:
                    player['Money'] += (player['Bet'] * 3) // 2
```

```
            print(
                "{}. You got Blackjack and receive {}¢!".format(player['Name'], ((player['Bet'] * 3) // 2) +
                                                                player['Bet']))
        if player['insurance'] and 0 < player.get('half_bet', 0):
            print("{} You lost your insurance bet".format(player['Name']))
            player['Money'] -= player['half_bet']
        elif Croupier['Score1'] > 21 and player['BJ'] is False:
            if player['Score1'] <= 21:
                player['Money'] += player['Bet']
                print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
            if player.get('Score2', 22) <= 21:
                player['Money'] += player['Bet']
                print('{}. You win! You get {}¢ from hand #2.'.format(player['Name'], player['Bet'] * 2))
            if player['Score1'] > 21:
                player['Money'] -= player['Bet']
                print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
            if player.get('Score2', 0) > 21:
                player['Money'] -= player['Bet']
                print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
        elif Croupier['Score1'] <= 21 and player['BJ'] is False:
            if Croupier['Score1'] < player['Score1'] <= 21:
                player['Money'] += player['Bet']
                print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
            if Croupier['Score1'] < player.get('Score2', 0) <= 21:
                player['Money'] += player['Bet']
                print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
            if Croupier['Score1'] == player['Score1']:
                print("{}. It's a tie, you recover your bet.".format(player['Name']))
            if Croupier['Score1'] == player.get('Score2', 0):
                print("{}. It's a tie, you recover your bet from hand #2.".format(player['Name']))
            if player['Score1'] < Croupier['Score1']:
                player['Money'] -= player['Bet']
                print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
            if player.get('Score2', Croupier['Score1']) < Croupier['Score1']:
                player['Money'] -= player['Bet']
                print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
            if player['Score1'] > 21:
                player['Money'] -= player['Bet']
                print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
            if player.get('Score2', 0) > 21:
```

```
                    player['Money'] -= player['Bet']
                    print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
    goodbye()


def goodbye():
    for x in Players:
        if Players[x]['Money'] < Config['Minimum_bet'] and Players[x]['Active']:
            print("Sorry, {}. You don't have enough funds to cover minimum bet. You only have left {}¢.".format(
                Players[x]['Name'], Players[x]['Money']))
            Players[x]['Active'] = False
            print('Thanks for playing! Come back another day!')
    inactive = 0
    for n in Players:  # Reset all markers to default
        if Players[n]['Active'] is False:
            inactive += 1
            if inactive == Config['Number of players']:
                print('No active players left. Thanks for playing!')
                sys.exit()
    print('If any player would like to withdraw, please type your name. Leave it blank and we will continue with the '
          'next game.')
    out = input("Player: ")
    for x in range(1, Config['Number of players'] + 1):
        new_player = 'Player ' + str(x)
        if out == Players[new_player].get('Name') and Players[new_player]['Active']:
            print('Farewell {}.'.format(Players[new_player]['Name']))
            net = Players[new_player]['Money'] - Config['Money']
            print('Net worth: {}¢'.format(net))
            Players[new_player]['Active'] = False
            goodbye()
    if out == '':
        for n in Players:  # Reset all markers to default
            if Players[n]['Active']:  # If any player is Active, it will reset its markers
                Players[n]['Deck1'].clear()
                Players[n]['c_values1'].clear()
                Players[n]['Play'] = True
                Players[n]['Double'] = False
                Players[n]['Ace'] = False
                Players[n]['BJ'] = False
                Players[n]['insurance'] = False
                if Players[n].get('Deck2'):
```

```
                Players[n].pop('Deck2')
                Players[n].pop('c_values2')
                Players[n].pop('Score2')
        Croupier['Deck1'].clear()
        Croupier['c_values1'].clear()
        Croupier['Play'] = True
        Croupier['BJ'] = False
        Croupier['Ace'] = False
        betting()
    else:
        print('No player found with the name {}.'.format(out))
        goodbye()


def drawCard(quantity, player, deck, deck_value):  # draw x cards and add to a dict so can keep track of each card in ordr no to have repeated card
    if Config['deck_size'] > (40 * Config['max_deck']):
        left = (12 * Config['max_deck'])
        print('\nOnly {} cards left in the deck!'.format(left))
        print('Time for reshuffling!')
        used_cards.clear()
        Config['deck_size'] = 0
    for x in range(quantity):
        while True:
            card_value = random.choice(list(Deck.keys()))
            card = card_value + ' ' + random.choice(Suit)
            if used_cards.get(card, 0) < Config['max_deck']:  # If card doesn't exit .get(card) = 0
                used_cards.setdefault(card, 0)  # Create card in dict used_cards with value = 0
                used_cards[card] += 1
                player[deck].append(card)
                player[deck_value].append(card_value)
                Config['deck_size'] += 1
                break

#=====================
#Graphical Design Interface UI

def cardVisualization(card):
    cardOutput = card[0]
    elem = []
```

```
    elem.clear()
    if "Hearts" in card:
        elem.append("♥")
    elif "Diamonds" in card:
        elem.append("♦")
    elif "Spades" in card:
        elem.append("♠")
    else:
        elem.append("♣")
    if "2" in cardOutput:
        elem.append("2")
    elif "3" in cardOutput:
        elem.append("3")
    elif "4" in cardOutput:
        elem.append("4")
    elif "5" in cardOutput:
        elem.append("5")
    elif "6" in cardOutput:
        elem.append("6")
    elif "7" in cardOutput:
        elem.append("7")
    elif "8" in cardOutput:
        elem.append("8")
    elif "9" in cardOutput:
        elem.append("9")
    elif "10" in cardOutput:
        elem.append("10")
    elif "Jack" in card:
        elem.append("J")
    elif "Queen" in card:
        elem.append("Q")
    elif "Ace" in card:
        elem.append("A")
    elif "King" in card:
        elem.append("K")
    else:
        elem.append("?")
    return elem


def print_deck(player, deck):
```

```
    for card in player[deck]:
        identifiers = cardVisualization(card)
        print(' ┌─────────┐ ')
        print(f'| {identifiers[0]}     |')
        print('|         |')
        print(f'|   {identifiers[1]}   |')
        print('|         |')
        print(f'|    {identifiers[0]}  |')
        print(' └─────────┘ ')

#===========
#Essential callable functions that determine the game

def valueCardSum(card_values): #Total value of player's deck
    sum_card = 0
    for y in card_values:
        sum_card += Deck.get(y)
    return sum_card

def show_card_value(player, deck, deck_value):
    print('Your cards are:')
    print_deck(player, deck)
    val = valueCardSum(player[deck_value])
    for card_v in player[deck_value]:
        if card_v == 'Ace':
            player['Ace'] = True
    if player['Ace'] and (val + 10) <= 21:
        print('There is an Ace. Possible values are:')
        print('Hard value: ' + str(val))
        print('Soft value: ' + str(val + 10))
    else:
        print('Its value is: {}'.format(val))


def hit_stand(player, deck, deck_value, score, state):
    while player['Active'] and player[state]:
        val = valueCardSum(player[deck_value])
        print('\n{}, it is your turn.'.format(player['Name']))
        show_card_value(player, deck, deck_value)
        if val > 21:
```

```
            print("Bust! Your hand is over 21.")
            player[score] = val
            player[state] = False
        elif val <= 21:
            while True:
                choice = input("What do you want to do, hit or stand? ")
                if choice not in ['hit', 'stand']:
                    print('Enter a valid input.')
                if choice == 'stand':
                    player[state] = False
                    if player['Ace'] and (val + 10) <= 21:
                        player[score] = val + 10
                    elif player['Ace'] and (val + 10) > 21:
                        player[score] = val
                    else:
                        player[score] = val
                    break
                elif choice == 'hit':
                    drawCard(1, player, deck, deck_value)
                    break


#Inspired by the program on Caseine and TP 5 that we show this
#It activates the object-ariented function that handles the flow of the game
if __name__ == "__main__":
    startGame() #Very important
```