Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3
«Шаблоны проектирования и модульное тестирование в Python»

Выполнил:
студент группы ИУ5-35Б
Ходырев Роман
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич
Подпись и дата:

Постановка задачи

1. Используя TDD-тестирование, BDD-тестирование и Mock-объекты, реализовать игру «BlackJack» с GUI-интерфейсом
2. При разработке использовать шаблон проектирования синглтон

Текст программы

BlackJack.py (основной файл)

```
from tkinter import *
from tkinter import ttk
import random
from copy import deepcopy
from PIL import Image, ImageTk
from logic import total
from read import read_singletone
from config_path import ICON_PATH, STAT_PATH, CARD_PATH, CROUPIER_PATH, RULE


#////////////////////////////ВЗЯТО ИЗ LOGIC.PY//////////////////////////
hearts = {
    "2_h": 2,
    "3_h": 3,
    "4_h": 4,
    "5_h": 5,
    "6_h": 6,
    "7_h": 7,
    "8_h": 8,
    "9_h": 9,
    "10_h": 10,
    "J_h": 10,
    "Q_h": 10,
    "K_h": 10,
    "A_h": 11
}

diamonds = {
    "2_d": 2,
    "3_d": 3,
    "4_d": 4,
    "5_d": 5,
    "6_d": 6,
    "7_d": 7,
    "8_d": 8,
    "9_d": 9,
    "10_d": 10,
    "J_d": 10,
    "Q_d": 10,
    "K_d": 10,
    "A_d": 11
}
```

```
spades = {
    "2_s": 2,
    "3_s": 3,
    "4_s": 4,
    "5_s": 5,
    "6_s": 6,
    "7_s": 7,
    "8_s": 8,
    "9_s": 9,
    "10_s": 10,
    "J_s": 10,
    "Q_s": 10,
    "K_s": 10,
    "A_s": 11
}

clubs = {
    "2_c": 2,
    "3_c": 3,
    "4_c": 4,
    "5_c": 5,
    "6_c": 6,
    "7_c": 7,
    "8_c": 8,
    "9_c": 9,
    "10_c": 10,
    "J_c": 10,
    "Q_c": 10,
    "K_c": 10,
    "A_c": 11
}

#Информация о всех картах
info = {}
info.update(hearts)
info.update(diamonds)
info.update(clubs)
info.update(spades)

#Непосредственно сами карты
all_cards = []
for m in [hearts, diamonds, spades, clubs]:
    all_cards.extend(m.keys())
#////////////////////////////////////////////////////////////////////////////


def finish():
    global widget
    widget.destroy()

def rules():
    global widget
    for w in widget.winfo_children():
        w.destroy()
```

```
    rule = Text(background="green", foreground="white", font=("Arial", 14), wrap="word")
    ys = ttk.Scrollbar(orient="vertical", command=rule.yview)

    rule.place(height=400, width=700, x=100, y=100)
    rule.insert("1.0", RULE)

    ys.pack(side=RIGHT, fill=Y)
    rule["yscrollcommand"] = ys.set

    back_button = ttk.Button(widget, text="Назад", command=exit_game)
    back_button.place(x=0, y=0)

def rewrite_stat():
    with open (STAT_PATH, 'w') as stat_file:
        stat_file.write("Games: 0\n")
        stat_file.write("Wons: 0\n")
        stat_file.write("Draws: 0\n")
        stat_file.write("Loses: 0\n")
        stat_file.write("Percant: 0")


def show_stat():
    global widget, stat_file
    games, wons, draws, loses, percent = stat_file.read_file()

    for w in widget.winfo_children():
        w.destroy()

    #Фрейм статистики и лэйблы
    stat_frame = ttk.Frame(borderwidth=1, relief=SOLID)
    games_label = ttk.Label(stat_frame, text=f"Количество игр: {games}",
background="green", foreground="white", font=("Arial", 16))
    wons_label = ttk.Label(stat_frame, text=f"Победы: {wons}", background="green",
foreground="white", font=("Arial", 16))
    draws_label = ttk.Label(stat_frame, text=f"Ничьи: {draws}", background="green",
foreground="white", font=("Arial", 16))
    loses_label = ttk.Label(stat_frame, text=f"Поражения: {loses}", background="green",
foreground="white", font=("Arial", 16))
    percent_label = ttk.Label(stat_frame, text=f"Процент побед: {percent}%",
background="green", foreground="white", font=("Arial", 16))

    games_label.pack(fill=X, ipadx=10, ipady=10)
    wons_label.pack(fill=X, ipadx=10, ipady=10)
    draws_label.pack(fill=X, ipadx=10, ipady=10)
    loses_label.pack(fill=X, ipadx=10, ipady=10)
    percent_label.pack(fill=X, ipadx=10, ipady=10)

    stat_frame.pack(expand=True)

    back_button = ttk.Button(widget, text="Назад", command=exit_game)
    back_button.place(x=0, y=0)
```

```
def start_game():
    global widget
    hide_widgets()
    game()

def hide_widgets():
    global widget
    for child_widget in widget.winfo_children():
        child_widget.destroy()

def show_widgets():
    global widget
    for child_widget in widget.winfo_children():
        child_widget.pack()

def exit_game():
    global widget
    for w in widget.winfo_children():
        w.destroy()
    main()

all_images = []
num_cards_on_screen = 0
def show_card(path, X, Y = 480):
    global widget
    global all_images
    global num_cards_on_screen
    image_file = Image.open(path)
    vp_image = ImageTk.PhotoImage(image_file)
    all_images.append(vp_image)

    label = Label(image=all_images[num_cards_on_screen])
    num_cards_on_screen += 1
    label.place(x=X, y=Y)
    #widget.update()
    #widget.update_idletasks()

def take_card(player, cards):
    global widget, stat_file
    player.append(cards[0])

    show_card(CARD_PATH.format(cards.pop(0)), 240+len(player)*80)

    player_tot = total(player)
    score = ttk.Label(widget, text=f"Сумма: {player_tot}", background="green",
foreground="white", font=("Arial", 14))
    score.place(height=40, width=100, x=10, y=520)

    if player_tot>21:
        stat_file.update_stat(-1)
        for w in widget.winfo_children():
            if type(w)==ttk.Button:
                w.destroy()
```

```
        end = ttk.Label(widget, text="Вы проиграли!", background="green",
foreground="white", font=("Arial", 24))
        end.place(height=90, width=240, x=350, y=250)
        menu_button = ttk.Button(text="Меню", command=main)
        new_game_button = ttk.Button(text="Новая игра", command=start_game)

        menu_button.place(height=50, width=120, x=340, y=330)
        new_game_button.place(height=50, width=120, x=470, y=330)

def croupier_take(croupie, cards, player):
    global widget, stat_file
    show_card(CARD_PATH.format(croupie[1]), 400, 150)
    player_tot = total(player)
    for w in widget.winfo_children():
        if type(w)==ttk.Button:
            w.destroy()

    croupie_tot = total(croupie)

    while croupie_tot<17:
        croupie.append(cards[0])
        show_card(CARD_PATH.format(cards.pop(0)), 240+len(croupie)*80, 150)
        croupie_tot = total(croupie)

    #После добора крупье считаем победителя
    if croupie_tot>21 or croupie_tot<player_tot:
        stat_file.update_stat(1)
        end = ttk.Label(widget, text="Вы выиграли!", background="green",
foreground="white", font=("Arial", 24))
        end.place(height=90, width=240, x=350, y=250)
    elif croupie_tot==player_tot:
        stat_file.update_stat(0)
        end = ttk.Label(widget, text="Ничья!", background="green", foreground="white",
font=("Arial", 24))
        end.place(height=90, width=240, x=350, y=250)
    else:
        stat_file.update_stat(-1)
        end = ttk.Label(widget, text="Вы проиграли!", background="green",
foreground="white", font=("Arial", 24))
        end.place(height=90, width=240, x=350, y=250)

    #Кнопки возврата в меню или начала новой игры
    menu_button = ttk.Button(text="Меню", command=main)
    new_game_button = ttk.Button(text="Новая игра", command=start_game)
    menu_button.place(height=50, width=120, x=340, y=330)
    new_game_button.place(height=50, width=120, x=470, y=330)


def game():

    global all_images
    global num_cards_on_screen
    all_images.clear()      #Очистка буфера после предыдущей игры
    num_cards_on_screen=0   #Очистка буфера после предыдущей игры
```

```
    X = 320
    Y = 480

    #Фото крупье
    croupier_file = Image.open(CROUPIER_PATH)
    vp_croupier = ImageTk.PhotoImage(croupier_file)
    all_images.append(vp_croupier)
    croupier_label = Label(background="green", image=all_images[num_cards_on_screen])
    num_cards_on_screen += 1
    croupier_label.place(height=120, width=140, x=350, y=10)

    #Выход из игры
    back_button = ttk.Button(widget, text="Выйти из игры", command=exit_game)
    back_button.pack(anchor='nw')

    #Тасовка карт
    random.shuffle(all_cards)
    cards = deepcopy(all_cards)

    #Раздаем начальные карты
    player = [cards[0], cards[1]]
    croupier = [cards[2], cards[3]]
    for i in range(4):
        cards.pop(0)

    player_tot = total(player)

    #Показываем начальные карты
    show_card(CARD_PATH.format(player[0]), X, Y)
    show_card(CARD_PATH.format(player[1]), X+80, Y)

    show_card(CARD_PATH.format(croupier[0]), X, 150)
    show_card(CARD_PATH.format("back"), X+80, 150)

    #Счет
    score = ttk.Label(widget, text=f"Сумма: {player_tot}", background="green",
foreground="white", font=("Arial", 14))
    score.place(height=40, width=100, x=10, y=520)

    #Передаем ход игроку
    new_card_button = ttk.Button(widget, text="Ещё!", command=lambda pl = player, cs =
cards: take_card(pl, cs))
    new_card_button.place(x=400, y=450)

    #Передаем ход крупье
    stop_card_button = ttk.Button(widget, text="Хватит", command=lambda cr = croupier, cs
= cards, pl = player: croupier_take(cr, cs, pl))
    stop_card_button.place(x=500, y=450)


def main():
    for w in widget.winfo_children():
        w.destroy()
```

```
    #Управление на главном окне
    frame_main = ttk.Frame(borderwidth=1, relief=SOLID)
    button_start = ttk.Button(frame_main, text="Новая игра", command = start_game,
width=50)
    button_continue = ttk.Button(frame_main, text="Правила", command=rules, width=50)
    button_stat = ttk.Button(frame_main, text="Статистика", command = show_stat, width=50)
    button_start.pack(fill=X, ipadx=10, ipady=10)
    button_continue.pack(fill=X, ipadx=10, ipady=10)
    button_stat.pack(fill=X, ipadx=10, ipady=10)

    #frame_main.place(x=300, y=232, height=136, width=300)
    frame_main.pack(expand=True)

    widget.mainloop()
    #widget.protocol("WM_DELETE__WINDOW", finish)


if __name__ == "__main__":
    widget = Tk()
    widget["bg"] = "green"
    widget.resizable(False, False)
    widget.title("BlackJack")
    icon = PhotoImage(file=ICON_PATH)
    widget.iconphoto(False, icon)
    widget.geometry("900x600+300+100")
    rewrite_stat()
    stat_file = read_singletone(STAT_PATH) #Переменная для работы со статистикой
    main()
```

config_path.py (файл конфигурации)

```
ICON_PATH = "images\\icon_cards.png"
BACK_BUTTON_PATH = "images\\go_back.png"
STAT_PATH = "stat.txt"
CARD_PATH = "images\\cards\\{}.jpg"
CROUPIER_PATH = "images\\croupier.png"

RULE = """Простая, динамичная и захватывающая игра. Игрок делает ставки на один или
несколько обозначенных на столе боксов.
После слов дилера «Ставок больше нет» и жеста закрытия ставок рукой, игроки не могут
трогать и менять свои ставки.

Вначале игрок получает две карты «в открытую», сумма очков которых позволяет решить, нужны
ему дополнительные карты или нет.
Цель игры – набрать 21 очко или близкую к этому сумму. Если игрок набирает сумму очков,
превышающую 21, то его ставка проигрывает.
Если сумма очков на картах дилера больше, чем 21, то все ставки, оставшиеся в игре,
выигрывают.

Игроки, набравшие сумму очков большую, чем дилер, выигрывают, их ставки оплачиваются 1:1.
Игроки, набравшие сумму очков меньшую, чем дилер, проигрывают.
Если сумма очков игрока равна сумме очков дилера, то объявляется «ничья» или Stay: ставка
игрока не выигрывает и не проигрывает.
```

```
Дилер набирает карты последним, при этом он обязан брать карту, если у него 16 очков или меньше, и остановиться, если сумма очков 17 или больше.
Тузы считаются как 1 или как 11, «картинки» (валеты, дамы и короли) — все по 10 очков, остальные карты соответствуют своему номиналу."""
```

read.py (синглтон для чтения текстовых данных)

```python
class read_singletone:
    _instance = None

    def __new__(cls, filename):
        if cls._instance is None:
            cls._instance = super(read_singletone, cls).__new__(cls)
            cls._instance.filename = filename
        return cls._instance

    def read_file(self):
        with open(self.filename, 'r') as stat_file:
            info_numbers = []
            for line in stat_file.readlines():
                info_numbers.append(line.split(": ")[1])
        games, wons, draws, loses, percant = info_numbers
        good_open = True

        #Преобразование типов и печать об удачном/неудачном открытии
        try:
            games = int(games)
            wons = int(wons)
            draws = int(draws)
            loses = int(loses)
            percant = float(percant)
        except:
            good_open = False

        if good_open:
            return games, wons, draws, loses, percant
        else:
            print("Ошибка!")

    def update_stat(self, result):
        games, wons, draws, loses, percent = self.read_file()
        games += 1
        if result == 1:
            wons += 1
        elif result == 0:
            draws += 1
        else:
            loses += 1
        percent = round((wons/games)*100, 2)
        with open (self.filename, 'w') as stat_file:
            stat_file.write(f"Games: {games}\n")
            stat_file.write(f"Wons: {wons}\n")
```

```
            stat_file.write(f"Draws: {draws}\n")
            stat_file.write(f"Loses: {loses}\n")
            stat_file.write(f"Percant: {percent}")
```

logic.py (логика игры в консоли)

```
#//////////////////РЕАЛИЗАЦИЯ ЛОГИКИ ИГРЫ В BLACKJACK//////////////////////////
import random
from copy import deepcopy

#Все карты и их значения
hearts = {
    "2_h": 2,
    "3_h": 3,
    "4_h": 4,
    "5_h": 5,
    "6_h": 6,
    "7_h": 7,
    "8_h": 8,
    "9_h": 9,
    "10_h": 10,
    "J_h": 10,
    "Q_h": 10,
    "K_h": 10,
    "A_h": 11
}

diamonds = {
    "2_d": 2,
    "3_d": 3,
    "4_d": 4,
    "5_d": 5,
    "6_d": 6,
    "7_d": 7,
    "8_d": 8,
    "9_d": 9,
    "10_d": 10,
    "J_d": 10,
    "Q_d": 10,
    "K_d": 10,
    "A_d": 11
}

spades = {
    "2_s": 2,
    "3_s": 3,
    "4_s": 4,
    "5_s": 5,
    "6_s": 6,
    "7_s": 7,
    "8_s": 8,
    "9_s": 9,
    "10_s": 10,
    "J_s": 10,
```

```
    "Q_s": 10,
    "K_s": 10,
    "A_s": 11
}

clubs = {
    "2_c": 2,
    "3_c": 3,
    "4_c": 4,
    "5_c": 5,
    "6_c": 6,
    "7_c": 7,
    "8_c": 8,
    "9_c": 9,
    "10_c": 10,
    "J_c": 10,
    "Q_c": 10,
    "K_c": 10,
    "A_c": 11
}

#Информация о всех картах
info = {}
info.update(hearts)
info.update(diamonds)
info.update(clubs)
info.update(spades)

#Непосредственно сами карты
all_cards = []
for m in [hearts, diamonds, spades, clubs]:
    all_cards.extend(m.keys())

#Подсчет суммы карт
def total(person):
    total_value = 0
    ace_count = 0

    for card in person:
        if card[0] in ['K', 'Q', 'J']:
            total_value += info[card]
        elif card[0] == 'A':
            ace_count += 1
            total_value += info[card]
        else:
            total_value += info[card]

    # Обработка того, что Ace может считаться за 1, если сумма больше 21
    while ace_count > 0 and total_value > 21:
        total_value -= 10
        ace_count -= 1

    return total_value
```

```python
#Покаазть карты
def show_cards(person, is_player=True):
    print("Ваши карты: " if is_player else "Карты крупье: ", end="")
    for card in person:
        print(card, end=" ")
    print()

#Определение победителя
def get_winner(p, c):
    if p>c:
        print("Вы выиграли")
    elif p==c:
        print("Ничья")
    else:
        print("Вы проиграли")


def player_turn(player, cards):
    #Считаем количество взятых карт
    num = 0
    correct = 0 #Необходима для корректировки значения туза - 1 или 11
    player_tot = total(player)
    show_cards(player)
    print("Сумма: ", player_tot)

    #Добор карт
    print("Взять еще карту? [y/n]")
    ans = input()
    while (ans!='n' and ans!='y'):
        print("Повторите ввод")
        ans = input()

    aces = {"A_s", "A_h", "A_c", "A_d"}
    while ans=='y':
        player.append(cards[num])
        num+=1
        player_tot = total(player) - correct
        show_cards(player)

        #Проверка на превышение и возврат индикаторного значения
        if player_tot>21:
            #При превышении туз считается не за 11 а за 1
            double = set(player).intersection(aces)              #Если у игрока есть туз
            if len(double)!=0:                                   #Находим общего туза у
игрока и среди всех тузов
                correct += 10                                    #Прибавляем к correct 10
(теперь этот туз не 11 а 1)
                aces.difference_update(double)                   #Убираем туза из aces
            else:
                return player, player_tot, -1

        player_tot = total(player) - correct
        print("Сумма: ", player_tot)
```

```
        print("Взять еще карту? [y/n]")
        ans = input()
        while (ans!='n' and ans!='y'):
            print("Повторите ввод")
            ans = input()

    return player, player_tot, num


def croupier_turn(croupie, cards):
    num = 0
    correct = 0 #Необходима для корректировки значения туза - 1 или 11
    croupie_tot = total(croupie)

    aces = {"A_s", "A_h", "A_c", "A_d"}
    while croupie_tot<17:
        croupie.append(cards[num])
        num+=1
        croupie_tot = total(croupie) - correct

        if croupie_tot>21:
            #При превышении туз считается не за 11 а за 1
            double = set(croupie).intersection(aces)      #Если у крупье есть туз
            if len(double)!=0:                                    #Находим общего туза у
крупье и среди всех тузов
                correct += 10                             #Прибавляем к correct 10 (теперь
этот туз не 11 а 1)
                aces.difference_update(double)            #Убираем туза из aces
            else:
                return croupie, croupie_tot, -1

        croupie_tot = total(croupie) - correct

    return croupie, croupie_tot, num

def init_game():
    #Тасовка карт
    random.shuffle(all_cards)
    cards = deepcopy(all_cards)

    # Раздаем начальные карты
    player = [cards[0], cards[1]]
    croupier = [cards[2], cards[3]]
    for i in range(4):
        cards.pop(0)

    player_tot = total(player)
    croupier_tot = total(croupier)

    return player, croupier, cards, player_tot, croupier_tot

def main():
    player, croupier, cards, player_tot, croupier_tot = init_game()
```

```
    #Передаем ход игроку
    player, player_tot, player_took = player_turn(player, cards)
    if player_took ==-1:
        print("Сумма: ", player_tot)
        print("Вы проиграли!")
        return

    #Удаляем из колоды карты, которые взял игрок
    for _ in range(player_took):
        cards.pop(0)

    #Передаем ход крупье
    croupier, croupier_tot, croupier_took = croupier_turn(croupier, cards)
    if croupier_took == -1:
        print("**************")
        show_cards(player)
        print("Сумма: ", player_tot)
        show_cards(croupier, False)
        print("Сумма: ", croupier_tot)
        print("**************")
        print("Вы выиграли")
        return
    for _ in range(croupier_took):
        cards.pop(0)

    print("**************")
    show_cards(player)
    print("Сумма: ", player_tot)
    show_cards(croupier, False)
    print("Сумма: ", croupier_tot)
    print("**************")

    get_winner(player_tot, croupier_tot)



if __name__ == "__main__":
    ans = 'y'
    while ans != 'n':
        main()
        print("Начать новую игру? [y/n]")
        ans = input()
        while (ans!='y' and ans!='n'):
            print("Повторите ввод")
            ans = input()
```

stat.txt (файл статистики)

```
Games: 0
Wons: 0
Draws: 0
Loses: 0
Percant: 0
```

test_blackjack.py (TDD-тестирование функции подсчета очков)

```python
import unittest
from logic import total

#Тестирование функции подсчета очков
class TestTotalFunction(unittest.TestCase):

    def test_1(self):
        cards = ["3_s", "2_h"]
        result = total(cards)
        self.assertEqual(result, 5)

    def test_2(self):
        cards = ["10_d", "9_c", "8_h", "7_s", "K_d"]
        result = total(cards)
        self.assertEqual(result, 44)

    def test_3(self):
        cards = ["10_d", "9_c", "A_h"]
        result = total(cards)
        self.assertEqual(result, 20)

    def test_4(self):
        cards = ["2_d", "3_c", "A_s"]
        result = total(cards)
        self.assertEqual(result, 16)

    def test_5(self):
        cards = ["A_c", "A_h"]
        result = total(cards)
        self.assertEqual(result, 12)

    def test_6(self):
        cards = ["A_c", "5_h", "A_d"]
        result = total(cards)
        self.assertEqual(result, 17)



if __name__ == "__main__":
    unittest.main()
```

blackjack_steps.py (BDD-тестирование)

```python
from behave import given, when, then
from logic import init_game

@given("есть запущенное приложение")
def step_given_app_is_running(context):
    pass

@when("игрок выбирает \"Новая игра\"")
def step_when_player_starts_new_game(context):
    context.player, context.croupier, context.cards, context.player_tot,
context.croupier_tot = init_game()  # Имплементируйте эту функцию

@then("на столе должны быть 2 карты у игрока")
def step_then_two_cards_on_player_table(context):
    player_hand = context.player  # Получите текущую руку игрока (например, из глобальной
переменной)
    assert len(player_hand) == 2

@then("на столе должна быть 2 карты у крупье")
def step_then_two_cards_on_dealer_table(context):
    dealer_hand = context.croupier  # Получите текущую руку крупье (например, из
глобальной переменной)
    assert len(dealer_hand) == 2

@then("сумма очков у игрока должна быть меньше или равна 21")
def step_then_player_score_should_be_equal_or_less_than_21(context):
    player_score = context.player_tot  # Получите текущий счёт игрока (например, из
глобальной переменной)
    assert player_score <= 21

@then("сумма очков крупье также меньше или равна 21")
def step_then_croupier_score_is_also_equal_or_less_then_21(context):
    croupier_score = context.croupier_tot
    assert croupier_score <= 21
```

blackjack.feature (сценарий BDD-тестирования)

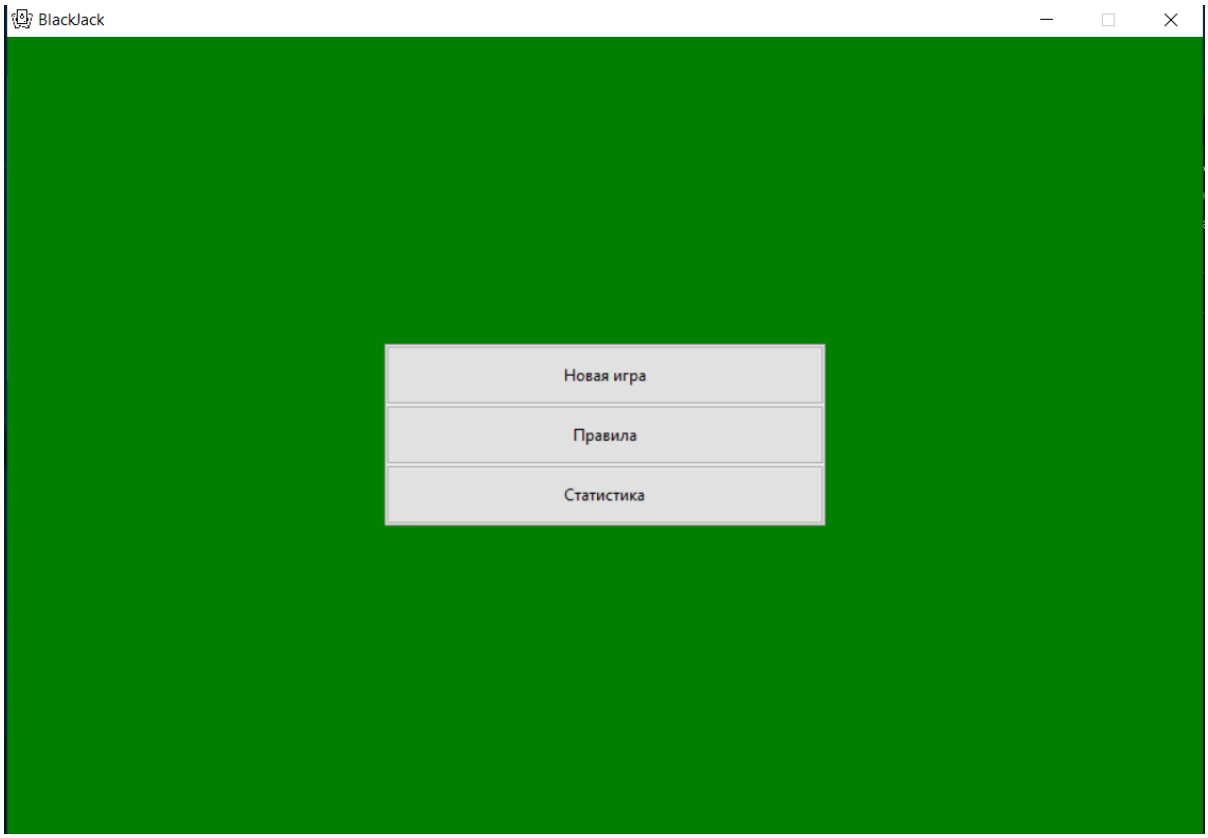
```
Feature: Новая игра в блэкджек

  Scenario: Запуск новой игры
    Given есть запущенное приложение
    When игрок выбирает "Новая игра"
    Then на столе должны быть 2 карты у игрока
    And на столе должна быть 2 карты у крупье
    And сумма очков у игрока должна быть меньше или равна 21
    And сумма очков крупье также меньше или равна 21
```

mock_object.py (mock-объект для тестирования чтения данных из файла)

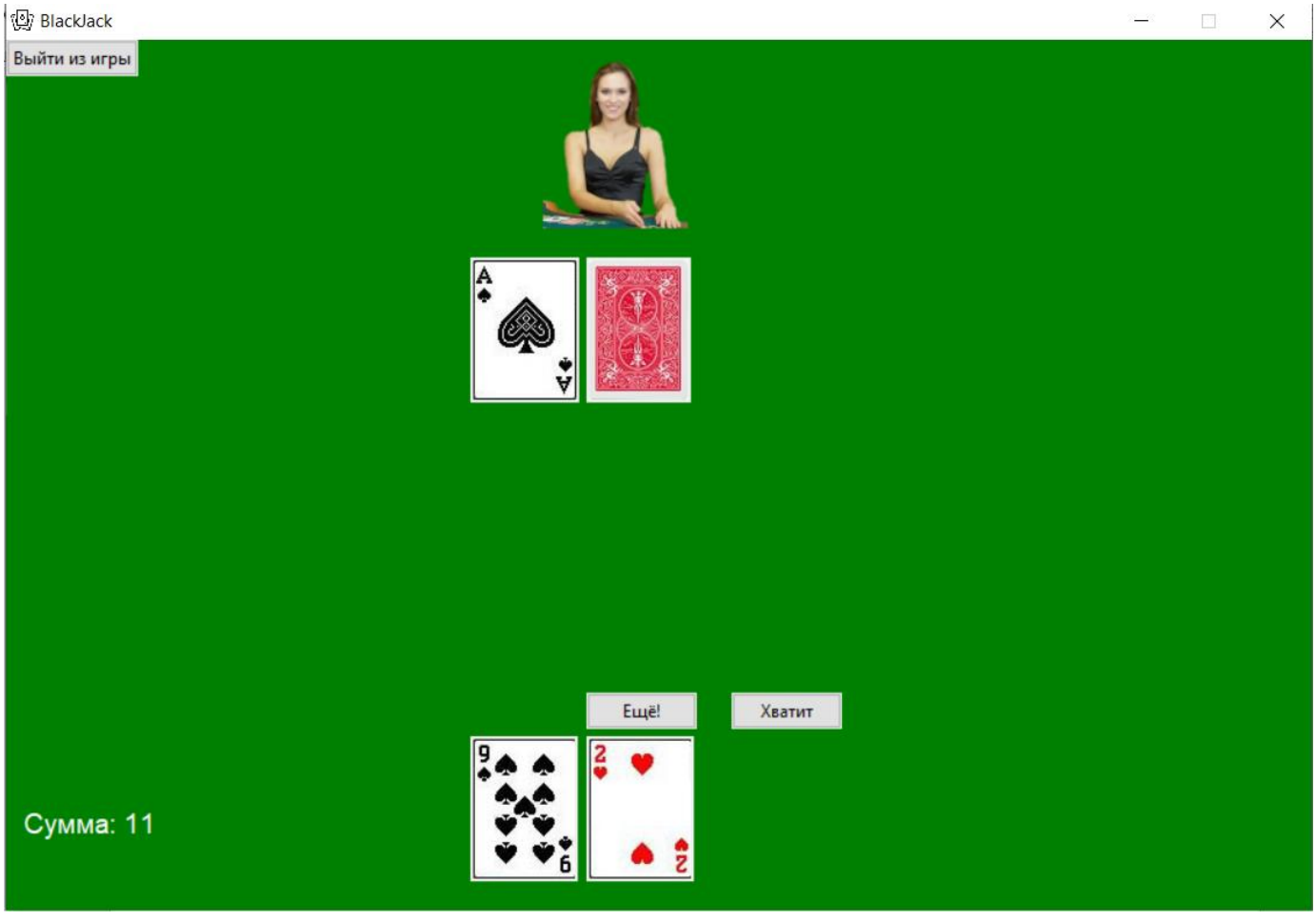
```
import unittest
from unittest.mock import patch
from read import read_singletone
from config_path import STAT_PATH

class TestReadSingletone(unittest.TestCase):
    @patch('builtins.open', unittest.mock.mock_open(read_data="Games: 0\nWons: 0\nDraws:
0\nLoses: 0\nPercant: 0.00"))
    def test_read_file(self):
        reader = read_singletone(STAT_PATH)
        games, wons, draws, loses, percent = reader.read_file()
        self.assertEqual(games, 0)
        self.assertEqual(wons, 0)
        self.assertEqual(draws, 0)
        self.assertEqual(loses, 0)
        self.assertEqual(percent, 0.00)

if __name__ == '__main__':
    unittest.main()
```

Анализ результатов

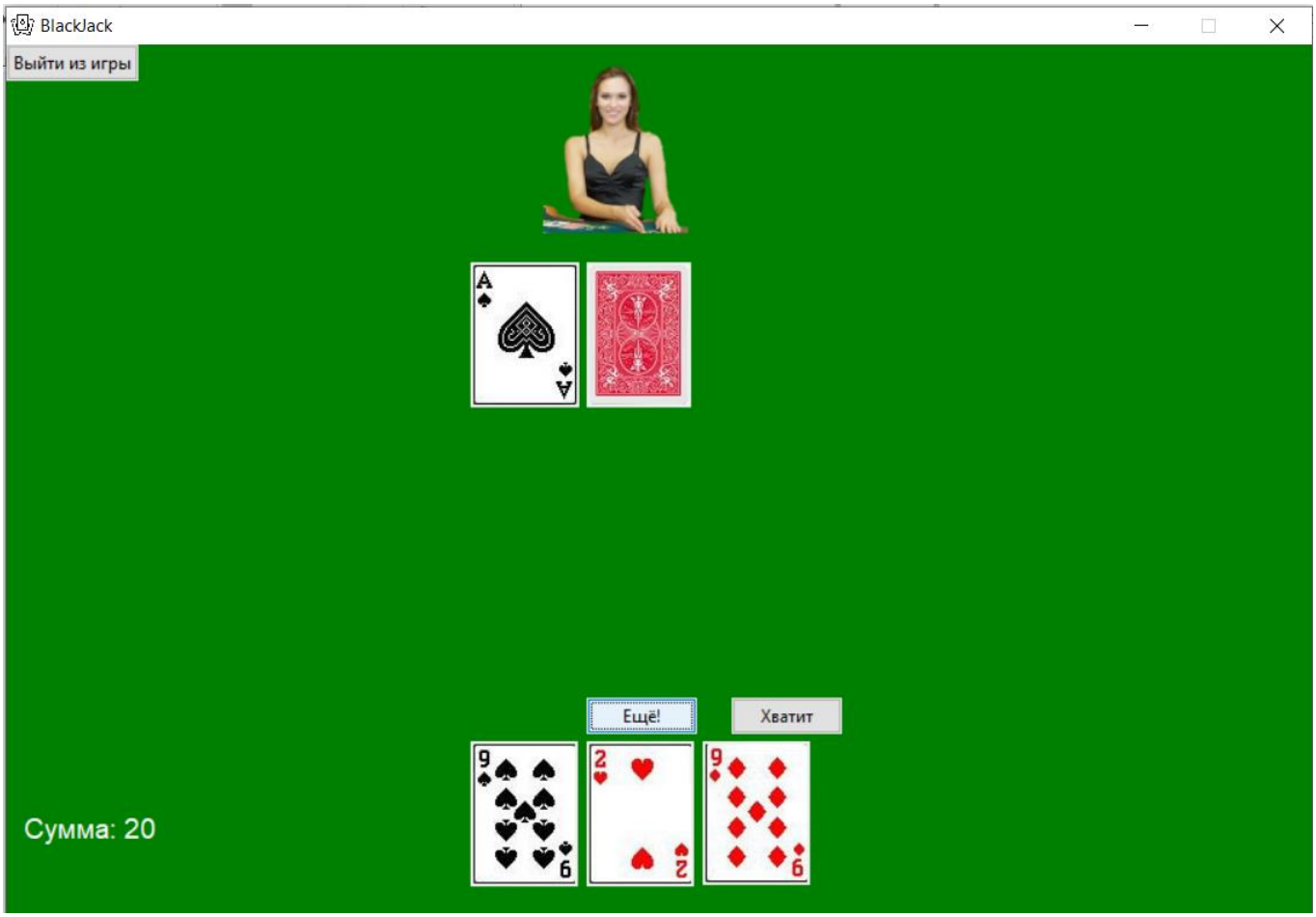
1) Меню игры



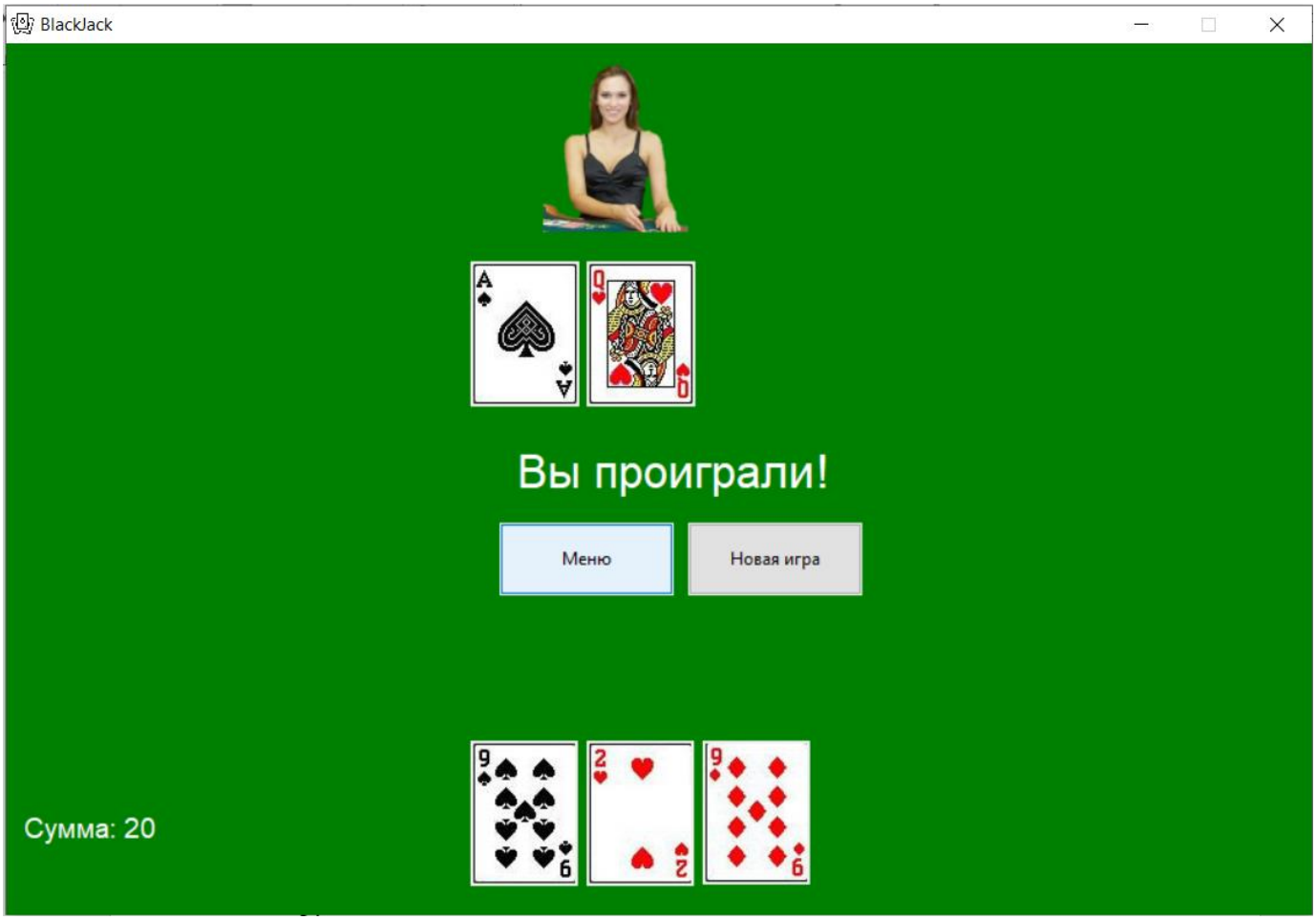
2) Новая игра



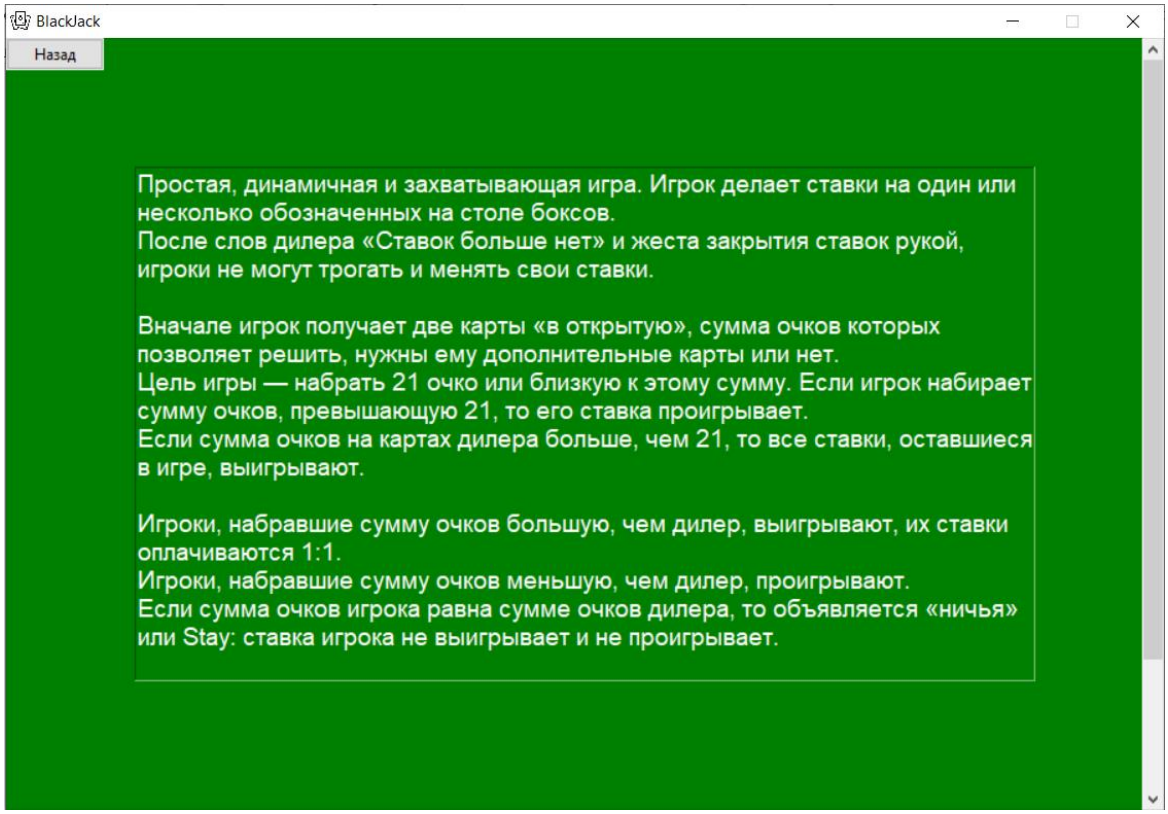
3) Взяли еще одну карту



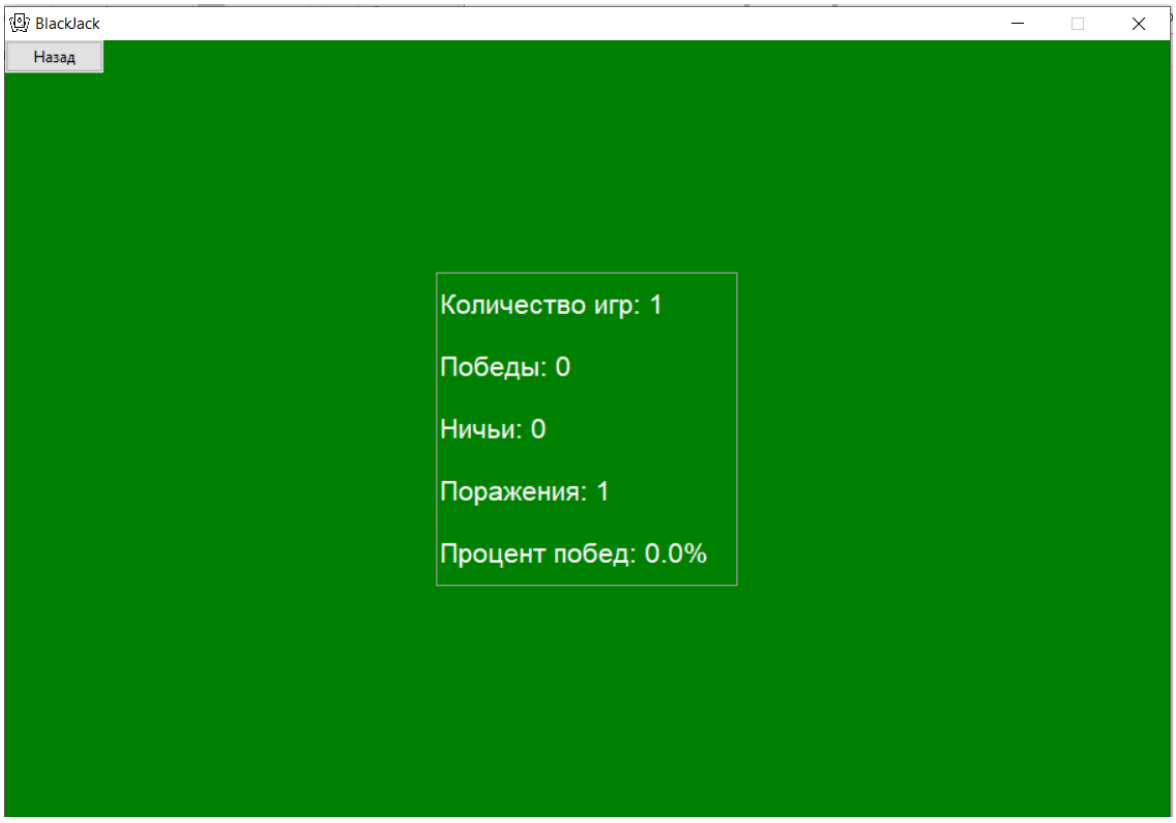
4) Хватит



5) Правила



6) Статистика



Вывод

Я на практике изучил методы, с помощью которых осуществляется модульное тестирование на языке Python и применил шаблон проектирования синглтон для создания своего проекта. У меня получилось создать игру для интересного времяпровождения