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 №4

Выполнил:
Студент группы ИУ5-33Б
Левкович Леонид

Проверил:
Преподаватель каф. ИУ5
Гапанюк Ю. Е.

Москва, 2025 г.

Задание:

Задание:

1. Разработайте простого бота для Telegram. Бот должен использовать функциональность создания кнопок.

Листинг программы:

```
import telebot
from telebot import types
import random
import json
import os
from datetime import datetime, timedelta
import time

# Токен вашего бота
BOT_TOKEN = '8451698197:AAGRh_3-t1HNxz4W7SG0Iv1l9TtGWUt8-nQ'
bot = telebot.TeleBot(BOT_TOKEN)

# Файл для хранения данных пользователей
DATA_FILE = 'blackjack_users.json'

# Эмодзи для карт
SUITS = {'♠': 'spades', '♥': 'hearts', '♦': 'diamonds', '♣': 'clubs'}
CARD_VALUES = {
    'A': 11, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
    '8': 8, '9': 9, '10': 10, 'J': 10, 'Q': 10, 'K': 10
}


# Глобальная система управления колодами
class DeckManager:
    def __init__(self):
        self.main_deck = []
        self.discard_pile = []
        self.total_cards = 0
        self.deck_count = 0
        self.is_initial = True
        self.shuffle_point = 0
        self.initialize_decks()

    def create_single_deck(self):
        """Создает одну колоду из 52 карт"""
        deck = []
        for suit in SUITS.keys():
            for card in CARD_VALUES.keys():
                deck.append((card, suit))
        return deck

    def initialize_decks(self):
        """Инициализация колод при запуске бота"""
        if self.is_initial:
            # При первом запуске 3-5 колод
            self.deck_count = random.randint(3, 5)
            self.is_initial = False
        else:
            # После перемешивания 5-8 колод
            self.deck_count = random.randint(5, 8)

        self.main_deck = []
        self.discard_pile = []

        # Создаем нужное количество колод
        for _ in range(self.deck_count):
```

```
            self.main_deck.extend(self.create_single_deck())

        random.shuffle(self.main_deck)
        self.total_cards = len(self.main_deck)
        self.shuffle_point = self.total_cards // 2  # Точка перемешивания - половина колод

        print(f"🎴 Инициализировано {self.deck_count} колод ({self.total_cards} карт)")
        print(f"🔄 Перемешивание при {self.shuffle_point} оставшихся картах")

    def draw_card(self):
        """Взять карту из колоды"""
        if not self.main_deck:
            self.initialize_decks()

        card = self.main_deck.pop()

        # Проверяем, нужно ли перемешать колоды
        if len(self.main_deck) <= self.shuffle_point:
            self.reshuffle()

        return card

    def discard_cards(self, cards):
        """Отправить карты в сброс"""
        self.discard_pile.extend(cards)

    def reshuffle(self):
        """Перемешивание колод при достижении точки перемешивания"""
        print(f"🔄 Перемешивание колод! Осталось {len(self.main_deck)} карт")
        self.initialize_decks()

        # Отправляем уведомление всем активным играм
        notification = (f"🎴 Колоды перемешаны!\n"
                        f"Используется {self.deck_count} колод ({self.total_cards} карт)\n"
                        f"Следующее перемешивание при {self.shuffle_point} картах")
        return notification

    def get_deck_info(self):
        """Получить информацию о состоянии колод"""
        cards_used = self.total_cards - len(self.main_deck)
        percent_used = (cards_used / self.total_cards * 100) if self.total_cards > 0 else 0

        return {
            'deck_count': self.deck_count,
            'total_cards': self.total_cards,
            'cards_remaining': len(self.main_deck),
            'cards_used': cards_used,
            'percent_used': percent_used,
            'shuffle_point': self.shuffle_point,
            'discard_count': len(self.discard_pile)
        }


# Класс для управления данными пользователей
class UserManager:
    def __init__(self):
        self.users = self.load_users()

    def load_users(self):
        if os.path.exists(DATA_FILE):
            with open(DATA_FILE, 'r', encoding='utf-8') as f:
                return json.load(f)
```

```
        return {}

    def save_users(self):
        with open(DATA_FILE, 'w', encoding='utf-8') as f:
            json.dump(self.users, f, ensure_ascii=False, indent=2)

    def get_user(self, user_id):
        user_id = str(user_id)
        if user_id not in self.users:
            self.users[user_id] = {
                'balance': 10000,
                'games_played': 0,
                'games_won': 0,
                'games_lost': 0,
                'biggest_win': 0,
                'daily_bonus': None,
                'current_game': None,
                'achievements': [],
                'vip_level': 0,
                'math_solved': 0,
                'last_math_bonus': None
            }
            self.save_users()
        return self.users[user_id]

    def update_user(self, user_id, data):
        user_id = str(user_id)
        self.users[user_id] = data
        self.save_users()


# Класс для игры в BlackJack
class BlackJackGame:
    def __init__(self, deck_manager):
        self.deck_manager = deck_manager
        self.player_hand = []
        self.dealer_hand = []
        self.bet = 0
        self.is_active = False
        self.doubled_down = False
        self.dealer_has_blackjack = False
        self.player_has_blackjack = False

    def draw_card(self):
        """Взять карту из общей колоды"""
        return self.deck_manager.draw_card()

    def calculate_hand(self, hand):
        value = 0
        aces = 0
        for card, _ in hand:
            if card == 'A':
                aces += 1
            value += CARD_VALUES[card]

        while value > 21 and aces > 0:
            value -= 10
            aces -= 1

        return value

    def format_hand(self, hand, hide_first=False):
        if hide_first:
            return f"[?, {hand[1][0]}{hand[1][1]}]"
        return ' '.join([f"{card}{suit}" for card, suit in hand])

    def start_game(self, bet):
```

```
        self.bet = bet
        self.player_hand = [self.draw_card(), self.draw_card()]
        self.dealer_hand = [self.draw_card(), self.draw_card()]
        self.is_active = True
        self.doubled_down = False

        # Проверка на BlackJack
        player_value = self.calculate_hand(self.player_hand)
        dealer_value = self.calculate_hand(self.dealer_hand)

        if player_value == 21:
            self.player_has_blackjack = True
        if dealer_value == 21:
            self.dealer_has_blackjack = True

        return 'continue'

    def end_game(self):
        """Завершение игры - отправка карт в сброс"""
        all_cards = self.player_hand + self.dealer_hand
        self.deck_manager.discard_cards(all_cards)
        self.is_active = False


# Инициализация
deck_manager = DeckManager()
user_manager = UserManager()
games = {}


# Клавиатуры
def get_main_keyboard():
    keyboard = types.ReplyKeyboardMarkup(resize_keyboard=True)
    keyboard.add(types.KeyboardButton("🎰 Начать игру"))
    keyboard.add(types.KeyboardButton("💰 Баланс"), types.KeyboardButton("📊
Статистика"))
    keyboard.add(types.KeyboardButton("🎁 Ежедневный бонус"),
types.KeyboardButton("🏆 Достижения"))
    keyboard.add(types.KeyboardButton("🧮 Математика"), types.KeyboardButton("👑
VIP"))
    keyboard.add(types.KeyboardButton("📖 Правила"))
    return keyboard


def get_bet_keyboard(balance):
    keyboard = types.InlineKeyboardMarkup()
    bets = [100, 500, 1000, 2500, 5000]
    for bet in bets:
        if balance >= bet:
            keyboard.add(types.InlineKeyboardButton(f"💵 {bet}",
callback_data=f"bet_{bet}"))
    if balance >= 100:
        keyboard.add(types.InlineKeyboardButton("✏️ Своя ставка",
callback_data="custom_bet"))
    keyboard.add(types.InlineKeyboardButton("❌ Отмена", callback_data="cancel"))
    return keyboard


def get_game_keyboard(can_double=False, balance=0, bet=0, player_value=0):
    keyboard = types.InlineKeyboardMarkup()

    if player_value < 21:
        keyboard.add(types.InlineKeyboardButton("👆 Взять карту",
callback_data="hit"))

    keyboard.add(types.InlineKeyboardButton("✋ Стоп", callback_data="stand"))
```

```
    if can_double and balance >= bet and player_value < 21:
        keyboard.add(types.InlineKeyboardButton("x2 Удвоить",
callback_data="double"))

    return keyboard


# Обработчики команд
@bot.message_handler(commands=['start'])
def start(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    welcome_text = f"""
🎰 <b>Добро пожаловать в BlackJack Bot!</b> 🎰

Привет, {message.from_user.first_name}! ✊

💰 Ваш начальный баланс: <b>{user_data['balance']} монет</b>

🎯 Цель игры - набрать 21 очко или близкое к этому значение, но не больше!

Используйте кнопки меню для навигации.
Удачной игры! 🍀
"""

    bot.send_message(message.chat.id, welcome_text, parse_mode='HTML',
reply_markup=get_main_keyboard())


@bot.message_handler(func=lambda message: message.text == "🎰 Начать игру")
def start_game(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    if user_data['balance'] < 100:
        bot.send_message(message.chat.id,
                         "❌ Недостаточно средств!\n"
                         "Минимальная ставка: 100 монет\n\n"
                         "🌅 Попробуйте решить математическую задачу для
получения бонуса!",
                         reply_markup=types.InlineKeyboardMarkup().add(
                             types.InlineKeyboardButton("🧮 Решить задачу за 1000
монет",

callback_data="math_bonus_bankrupt")
                         ))
        return

    bot.send_message(message.chat.id,
                     f"💰 Ваш баланс: {user_data['balance']} монет\n"
                     "Выберите ставку:",
                     reply_markup=get_bet_keyboard(user_data['balance']))


@bot.message_handler(func=lambda message: message.text == "💰 Баланс")
def show_balance(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    vip_emoji = ["", "⭐", "🌟", "💫", "✨", "👑"][min(user_data['vip_level'],
5)]

    balance_text = f"""
```

```
💰 <b>Ваш баланс</b> 💰

Монеты: <b>{user_data['balance']}</b> 
VIP уровень: {user_data['vip_level']} {vip_emoji}

{
    "❗ Баланс низкий! Получите ежедневный бонус или решите задачу."
    if user_data['balance'] < 500 else
    "✅ Отличный баланс для игры!"
    }
"""

    bot.send_message(message.chat.id, balance_text, parse_mode='HTML')


@bot.message_handler(func=lambda message: message.text == "📊 Статистика")
def show_stats(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    win_rate = 0
    if user_data['games_played'] > 0:
        win_rate = (user_data['games_won'] / user_data['games_played']) * 100

    stats_text = f"""
📊 <b>Ваша статистика</b> 📊

🎮 Игр сыграно: {user_data['games_played']}
✅ Побед: {user_data['games_won']}
❌ Поражений: {user_data['games_lost']}
📈 Процент побед: {win_rate:.1f}%
💰 Крупнейший выигрыш: {user_data['biggest_win']} монет
 Решено задач: {user_data.get('math_solved', 0)}
🏆 Достижений: {len(user_data['achievements'])}
"""

    bot.send_message(message.chat.id, stats_text, parse_mode='HTML')


@bot.message_handler(func=lambda message: message.text == "🎁 Ежедневный бонус")
def daily_bonus(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    if user_data['daily_bonus']:
        last_bonus = datetime.fromisoformat(user_data['daily_bonus'])
        if datetime.now() - last_bonus < timedelta(hours=24):
            time_left = timedelta(hours=24) - (datetime.now() - last_bonus)
            hours = int(time_left.total_seconds() // 3600)
            minutes = int((time_left.total_seconds() % 3600) // 60)
            bot.send_message(message.chat.id,
                             f"⏰ Следующий бонус через: {hours}ч {minutes}мин")
            return

    bonus = random.randint(500, 2000) + (user_data['vip_level'] * 100)
    user_data['balance'] += bonus
    user_data['daily_bonus'] = datetime.now().isoformat()
    user_manager.update_user(user_id, user_data)

    bot.send_message(message.chat.id,
                     f"🎁 Вы получили ежедневный бонус: {bonus} монет!\n"
                     f"💰 Новый баланс: {user_data['balance']} монет")


@bot.message_handler(func=lambda message: message.text == " Математика")
```

```
def math_menu(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    can_solve = True
    time_left_text = ""

    if user_data.get('last_math_bonus'):
        last_math = datetime.fromisoformat(user_data['last_math_bonus'])
        if datetime.now() - last_math < timedelta(minutes=30):
            can_solve = False
            time_left = timedelta(minutes=30) - (datetime.now() - last_math)
            minutes = int(time_left.total_seconds() // 60)
            seconds = int(time_left.total_seconds() % 60)
            time_left_text = f"\n⏰ Следующая задача через: {minutes}м
{seconds}с"

    text = f"""
🧮 <b>Математические задачи</b> 🧮

Решайте задачи и получайте монеты!

📚 Решено задач: {user_data.get('math_solved', 0)}

Награды:
• Обычная задача: 200-500 монет
• При банкротстве: 1000 монет{time_left_text}
"""

    keyboard = types.InlineKeyboardMarkup()
    if can_solve:
        keyboard.add(types.InlineKeyboardButton("🎯 Решить задачу",
callback_data="math_bonus_regular"))
    if user_data['balance'] < 100:
        keyboard.add(types.InlineKeyboardButton("💸 Задача банкрота (1000
монет)", callback_data="math_bonus_bankrupt"))

    bot.send_message(message.chat.id, text, parse_mode='HTML',
reply_markup=keyboard)


@bot.message_handler(func=lambda message: message.text == "🏆 Достижения")
def show_achievements(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    all_achievements = {
        'first_win': '🏆 Первая победа',
        'win_streak_5': '🔥 5 побед подряд',
        'win_streak_10': '💥 10 побед подряд',
        'big_winner': '💰 Выигрыш 10000+',
        'lucky_seven': '7️⃣ Выиграть с 7-7-7',
        'blackjack_master': '🎯 10 BlackJack',
        'vip_player': '💎 VIP игрок',
        'mathematician': '🧮 Гений математики (50 задач)',
        'math_beginner': '📐 Начинающий математик (10 задач)',
        'shuffle_witness': '🔄 Свидетель перемешивания'
    }

    text = "🏆 <b>Ваши достижения</b> 🏆\n\n"

    for ach_id, ach_name in all_achievements.items():
        if ach_id in user_data['achievements']:
            text += f"✅ {ach_name}\n"
        else:
```

```
            text += f"🔗 {ach_name}\n"

    bot.send_message(message.chat.id, text, parse_mode='HTML')


@bot.message_handler(func=lambda message: message.text == "📖 Правила")
def show_rules(message):
    rules_text = """
📖 <b>Правила BlackJack</b> 📖

🎯 <b>Цель игры:</b>
Набрать 21 очко или близкое значение, не превысив его.

🏷️ <b>Значения карт:</b>
• 2-10 - номинал карты
• J, Q, K - 10 очков
• A (туз) - 11 или 1 очко

🎮 <b>Ход игры:</b>
1. Делаете ставку
2. Получаете 2 карты
3. Решаете: взять еще карту или остановиться
4. Дилер открывает свои карты
5. Побеждает тот, у кого больше очков (но не больше 21)

💰 <b>Выплаты:</b>
• Обычная победа: 2:1
• BlackJack (21 с двух карт): 2.5:1
• Ничья: возврат ставки

🃏 <b>Система колод:</b>
• Используется от 3 до 8 колод одновременно
• Карты перемешиваются при использовании половины
• Использованные карты уходят в сброс

🎁 <b>Бонусы:</b>
• Ежедневный бонус каждые 24 часа
• Математические задачи каждые 30 минут
• При банкротстве - особая задача на 1000 монет
• VIP уровни за активную игру
"""

    bot.send_message(message.chat.id, rules_text, parse_mode='HTML')


@bot.message_handler(func=lambda message: message.text == "👑 VIP")
def show_vip(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    vip_text = f"""
👑 <b>VIP Система</b> 👑

Ваш VIP уровень: {user_data['vip_level']} / 5

<b>Преимущества VIP:</b>
⭐ Уровень 1: +100 к ежедневному бонусу
🌟 Уровень 2: +200 к ежедневному бонусу
💫 Уровень 3: +300 к ежедневному бонусу
✨ Уровень 4: +400 к ежедневному бонусу
👑 Уровень 5: +500 к ежедневному бонусу + эксклюзивные достижения

<b>Как повысить уровень:</b>
• Играйте каждый день
```

```
• Выигрывайте крупные суммы
• Получайте достижения

До следующего уровня: {(user_data['vip_level'] + 1) * 10 -
user_data['games_played']} игр
"""

    bot.send_message(message.chat.id, vip_text, parse_mode='HTML')


# Обработчики callback
@bot.callback_query_handler(func=lambda call: call.data.startswith('bet_'))
def handle_bet(call):
    user_id = call.from_user.id
    bet_amount = int(call.data.split('_')[1])

    user_data = user_manager.get_user(user_id)

    if user_data['balance'] < bet_amount:
        bot.answer_callback_query(call.id, "❌ Недостаточно средств!")
        return

    # Создаем новую игру с общим менеджером колод
    game = BlackJackGame(deck_manager)
    games[user_id] = game

    # Снимаем ставку
    user_data['balance'] -= bet_amount

    # Начинаем игру
    game.start_game(bet_amount)

    player_value = game.calculate_hand(game.player_hand)

    # Формируем сообщение
    game_text = f"""
🎰 <b>Игра началась!</b> 🎰
Ставка: {bet_amount} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand, hide_first=True)}
"""

    # Если у игрока BlackJack
    if game.player_has_blackjack:
        game_text += "\n🎯 У вас BlackJack!"

        if game.dealer_has_blackjack:
            dealer_value = game.calculate_hand(game.dealer_hand)
            game_text = f"""
🎰 <b>Результаты</b> 🎰
Ставка: {bet_amount} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand)}
Сумма: <b>{dealer_value}</b>

🤝 <b>Ничья! Оба собрали BlackJack!</b>
Ставка возвращена.
"""
            user_data['balance'] += bet_amount
            user_data['games_played'] += 1
```

```
            game.end_game()  # Отправляем карты в сброс
        else:
            handle_player_blackjack(call, game, user_data, bet_amount)
            user_manager.update_user(user_id, user_data)
            bot.answer_callback_query(call.id)
            return

        bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id, parse_mode='HTML')
    else:
        can_double = len(game.player_hand) == 2
        keyboard = get_game_keyboard(can_double, user_data['balance'],
bet_amount, player_value)
        bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id,
                              parse_mode='HTML', reply_markup=keyboard)

    user_manager.update_user(user_id, user_data)
    bot.answer_callback_query(call.id)


def handle_player_blackjack(call, game, user_data, bet_amount):
    # Дилер открывает карты и играет по правилам
    dealer_value = game.calculate_hand(game.dealer_hand)

    while dealer_value < 17:
        game.dealer_hand.append(game.draw_card())
        dealer_value = game.calculate_hand(game.dealer_hand)

    player_value = 21  # У игрока BlackJack

    game_text = f"""
🎰 <b>Результаты</b> 🎰
Ставка: {bet_amount} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b> - BlackJack!

🎩 Карты дилера: {game.format_hand(game.dealer_hand)}
Сумма: <b>{dealer_value}</b>
"""

    if dealer_value > 21 or player_value > dealer_value:
        winnings = int(bet_amount * 2.5)
        user_data['balance'] += winnings
        user_data['games_won'] += 1
        user_data['games_played'] += 1
        if winnings > user_data['biggest_win']:
            user_data['biggest_win'] = winnings
        game_text += f"\n🎉 <b>BlackJack! Вы выиграли!</b>\n+{winnings} монет!"
        check_achievements(call.from_user.id, user_data, 'blackjack')

    game.end_game()  # Отправляем карты в сброс
    bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id, parse_mode='HTML')


@bot.callback_query_handler(func=lambda call: call.data == 'hit')
def handle_hit(call):
    user_id = call.from_user.id

    if user_id not in games or not games[user_id].is_active:
        bot.answer_callback_query(call.id, "❌ Игра не найдена!")
        return

    game = games[user_id]
    game.player_hand.append(game.draw_card())
```

```
    player_value = game.calculate_hand(game.player_hand)

    game_text = f"""
🎰 <b>Игра продолжается</b> 🎰
Ставка: {game.bet} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand, hide_first=True)}
"""

    if player_value > 21:
        # Перебор
        game.is_active = False
        user_data = user_manager.get_user(user_id)
        user_data['games_played'] += 1
        user_data['games_lost'] += 1
        user_manager.update_user(user_id, user_data)

        dealer_value = game.calculate_hand(game.dealer_hand)
        game_text = f"""
🎰 <b>Игра окончена</b> 🎰
Ставка: {game.bet} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand)}
Сумма: <b>{dealer_value}</b>

💥 <b>Перебор!</b>
Вы проиграли {game.bet} монет!
"""
        game.end_game()  # Отправляем карты в сброс
        bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id, parse_mode='HTML')

    elif player_value == 21:
        handle_stand_logic(call, auto=True)

    else:
        keyboard = get_game_keyboard(False, player_value=player_value)
        bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id,
                              parse_mode='HTML', reply_markup=keyboard)

    bot.answer_callback_query(call.id)


@bot.callback_query_handler(func=lambda call: call.data == 'stand')
def handle_stand(call):
    handle_stand_logic(call)
    bot.answer_callback_query(call.id)


def handle_stand_logic(call, auto=False):
    user_id = call.from_user.id

    if user_id not in games or not games[user_id].is_active:
        if not auto:
            bot.answer_callback_query(call.id, "❌ Игра не найдена!")
        return

    game = games[user_id]
    game.is_active = False
```

```
    player_value = game.calculate_hand(game.player_hand)
    dealer_value = game.calculate_hand(game.dealer_hand)

    while dealer_value < 17:
        game.dealer_hand.append(game.draw_card())
        dealer_value = game.calculate_hand(game.dealer_hand)

    game_text = f"""
🎰 <b>Результаты</b> 🎰
Ставка: {game.bet} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand)}
Сумма: <b>{dealer_value}</b>
"""

    user_data = user_manager.get_user(user_id)
    user_data['games_played'] += 1

    # Определяем победителя
    if dealer_value > 21:
        winnings = game.bet * 2
        if game.doubled_down:
            winnings *= 2
        user_data['balance'] += winnings
        user_data['games_won'] += 1
        if winnings > user_data['biggest_win']:
            user_data['biggest_win'] = winnings
        game_text += f"\n🎉 <b>Дилер перебрал! Вы выиграли!</b>\n+{winnings}
монет!"
        check_achievements(user_id, user_data, 'win')

    elif player_value > dealer_value:
        winnings = game.bet * 2
        if game.doubled_down:
            winnings *= 2
        user_data['balance'] += winnings
        user_data['games_won'] += 1
        if winnings > user_data['biggest_win']:
            user_data['biggest_win'] = winnings
        game_text += f"\n🎉 <b>Вы выиграли!</b>\n+{winnings} монет!"
        check_achievements(user_id, user_data, 'win')

    elif player_value < dealer_value:
        user_data['games_lost'] += 1
        loss = game.bet
        if game.doubled_down:
            loss *= 2
        game_text += f"\n😔 <b>Вы проиграли!</b>\n-{loss} монет"

    else:
        refund = game.bet
        if game.doubled_down:
            refund *= 2
        user_data['balance'] += refund
        game_text += f"\n🤝 <b>Ничья!</b>\nСтавка возвращена."

    # Проверяем, было ли перемешивание
    deck_info = deck_manager.get_deck_info()
    if deck_info['cards_remaining'] <= deck_info['shuffle_point']:
        if 'shuffle_witness' not in user_data['achievements']:
            user_data['achievements'].append('shuffle_witness')
            game_text += "\n\n🏆 Получено достижение: Свидетель перемешивания!"
```

```
    # Обновляем VIP уровень
    if user_data['games_played'] >= (user_data['vip_level'] + 1) * 10:
        user_data['vip_level'] = min(5, user_data['vip_level'] + 1)
        game_text += f"\n\n🎊 Поздравляем! VIP уровень повышен до
{user_data['vip_level']}!"

    game.end_game()  # Отправляем карты в сброс
    user_manager.update_user(user_id, user_data)
    bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id, parse_mode='HTML')


@bot.callback_query_handler(func=lambda call: call.data == 'double')
def handle_double(call):
    user_id = call.from_user.id

    if user_id not in games or not games[user_id].is_active:
        bot.answer_callback_query(call.id, "❌ Игра не найдена!")
        return

    game = games[user_id]
    user_data = user_manager.get_user(user_id)

    if user_data['balance'] < game.bet:
        bot.answer_callback_query(call.id, "❌ Недостаточно средств для
удвоения!")
        return

    user_data['balance'] -= game.bet
    game.doubled_down = True

    game.player_hand.append(game.draw_card())
    player_value = game.calculate_hand(game.player_hand)

    if player_value > 21:
        game.is_active = False
        user_data['games_played'] += 1
        user_data['games_lost'] += 1
        user_manager.update_user(user_id, user_data)

        dealer_value = game.calculate_hand(game.dealer_hand)
        game_text = f"""
🎰 <b>Удвоение - Перебор!</b> 🎰
Ставка: {game.bet * 2} монет

👤 Ваши карты: {game.format_hand(game.player_hand)}
Сумма: <b>{player_value}</b>

🎩 Карты дилера: {game.format_hand(game.dealer_hand)}
Сумма: <b>{dealer_value}</b>

💥 <b>Перебор!</b>
Вы проиграли {game.bet * 2} монет!
"""
        game.end_game()  # Отправляем карты в сброс
        bot.edit_message_text(game_text, call.message.chat.id,
call.message.message_id, parse_mode='HTML')
    else:
        handle_stand_logic(call, auto=True)

    bot.answer_callback_query(call.id)


@bot.callback_query_handler(func=lambda call:
call.data.startswith('math_bonus'))
```

```
def math_bonus(call):
    user_id = call.from_user.id
    user_data = user_manager.get_user(user_id)

    is_bankrupt = call.data == 'math_bonus_bankrupt'

    if not is_bankrupt and user_data.get('last_math_bonus'):
        last_math = datetime.fromisoformat(user_data['last_math_bonus'])
        if datetime.now() - last_math < timedelta(minutes=30):
            time_left = timedelta(minutes=30) - (datetime.now() - last_math)
            minutes = int(time_left.total_seconds() // 60)
            bot.answer_callback_query(call.id, f"⏳ Подождите еще {minutes}
минут!")
            return

    # Генерируем математический пример
    if is_bankrupt:
        operations = ['+', '-']
        operation = random.choice(operations)

        if operation == '+':
            a = random.randint(10, 50)
            b = random.randint(10, 50)
            answer = a + b
            question = f"{a} + {b}"
        else:
            a = random.randint(30, 70)
            b = random.randint(10, 29)
            answer = a - b
            question = f"{a} - {b}"

        reward = 1000
    else:
        operations = ['+', '-', '*']
        operation = random.choice(operations)

        if operation == '+':
            a = random.randint(20, 99)
            b = random.randint(20, 99)
            answer = a + b
            question = f"{a} + {b}"
        elif operation == '-':
            a = random.randint(50, 99)
            b = random.randint(10, 49)
            answer = a - b
            question = f"{a} - {b}"
        else:
            a = random.randint(7, 15)
            b = random.randint(7, 15)
            answer = a * b
            question = f"{a} × {b}"

        reward = random.randint(200, 500)

    user_data['math_answer'] = answer
    user_data['math_attempts'] = 3
    user_data['math_reward'] = reward
    user_data['math_is_bankrupt'] = is_bankrupt
    user_manager.update_user(user_id, user_data)

    text = f"""
🧮 <b>Математическая задача</b> 🧮

Решите пример для получения {reward} монет:

<b>{question} = ?</b>
```

```
У вас есть 3 попытки.
Отправьте ответ в чат.
"""

    bot.send_message(call.message.chat.id, text, parse_mode='HTML')
    bot.answer_callback_query(call.id)


@bot.callback_query_handler(func=lambda call: call.data == 'custom_bet')
def custom_bet(call):
    bot.send_message(call.message.chat.id,
                     "💵 Введите сумму ставки (минимум 100 монет):")
    bot.register_next_step_handler(call.message, process_custom_bet)
    bot.answer_callback_query(call.id)


def process_custom_bet(message):
    try:
        bet_amount = int(message.text)
        if bet_amount < 100:
            bot.send_message(message.chat.id, "❌ Минимальная ставка 100 монет!")
            return

        user_id = message.from_user.id
        user_data = user_manager.get_user(user_id)

        if user_data['balance'] < bet_amount:
            bot.send_message(message.chat.id, "❌ Недостаточно средств!")
            return

        class FakeCall:
            def __init__(self, user_id, chat_id, message_id, data):
                self.from_user = type('obj', (object,), {'id': user_id})
                self.message = type('obj', (object,),
                                    {'chat': type('obj', (object,), {'id':
chat_id}), 'message_id': message_id})
                self.data = data
                self.id = str(random.randint(1000, 9999))

        fake_call = FakeCall(user_id, message.chat.id, message.message_id + 1,
f'bet_{bet_amount}')

        sent = bot.send_message(message.chat.id, "Обработка...")
        fake_call.message.message_id = sent.message_id

        handle_bet(fake_call)

    except ValueError:
        bot.send_message(message.chat.id, "❌ Введите корректное число!")


@bot.callback_query_handler(func=lambda call: call.data == 'cancel')
def cancel_game(call):
    bot.edit_message_text("❌ Игра отменена", call.message.chat.id,
call.message.message_id)
    bot.answer_callback_query(call.id)


@bot.message_handler(func=lambda message: True)
def handle_math_answer(message):
    user_id = message.from_user.id
    user_data = user_manager.get_user(user_id)

    if 'math_answer' in user_data and user_data['math_attempts'] > 0:
        try:
            answer = int(message.text)
            if answer == user_data['math_answer']:
```

```
                reward = user_data['math_reward']
                user_data['balance'] += reward
                user_data['math_solved'] = user_data.get('math_solved', 0) + 1

                if not user_data.get('math_is_bankrupt', False):
                    user_data['last_math_bonus'] = datetime.now().isoformat()

                del user_data['math_answer']
                del user_data['math_attempts']
                del user_data['math_reward']
                if 'math_is_bankrupt' in user_data:
                    del user_data['math_is_bankrupt']

                user_manager.update_user(user_id, user_data)

                bot.send_message(message.chat.id,
                                 f"✔ Правильно! Вы получили {reward} монет!\n"
                                 f"💰 Новый баланс: {user_data['balance']}
монет\n"
                                 f"📚 Всего решено задач:
{user_data['math_solved']}")

                if user_data['math_solved'] == 10 and 'math_beginner' not in
user_data['achievements']:
                    user_data['achievements'].append('math_beginner')
                    user_manager.update_user(user_id, user_data)
                    bot.send_message(message.chat.id, "🏆 Получено достижение:
Начинающий математик!")
                elif user_data['math_solved'] == 50 and 'mathematician' not in
user_data['achievements']:
                    user_data['achievements'].append('mathematician')
                    user_manager.update_user(user_id, user_data)
                    bot.send_message(message.chat.id, "🏆 Получено достижение:
Гений математики!")
            else:
                user_data['math_attempts'] -= 1
                user_manager.update_user(user_id, user_data)

                if user_data['math_attempts'] > 0:
                    bot.send_message(message.chat.id,
                                     f"✖ Неправильно! Осталось попыток:
{user_data['math_attempts']}")
                else:
                    del user_data['math_answer']
                    del user_data['math_attempts']
                    del user_data['math_reward']
                    if 'math_is_bankrupt' in user_data:
                        del user_data['math_is_bankrupt']
                    user_manager.update_user(user_id, user_data)
                    bot.send_message(message.chat.id, "😔 Попытки закончились.
Попробуйте позже!")
        except ValueError:
            pass


def check_achievements(user_id, user_data, event_type):
    new_achievements = []

    if event_type == 'win' and user_data['games_won'] == 1 and 'first_win' not
in user_data['achievements']:
        user_data['achievements'].append('first_win')
        new_achievements.append('🏆 Первая победа')

    if event_type == 'blackjack':
        if 'blackjack_count' not in user_data:
            user_data['blackjack_count'] = 0
```

```
        user_data['blackjack_count'] += 1

        if user_data['blackjack_count'] >= 10 and 'blackjack_master' not in
user_data['achievements']:
            user_data['achievements'].append('blackjack_master')
            new_achievements.append('🎯 10 BlackJack')

    if user_data['biggest_win'] >= 10000 and 'big_winner' not in
user_data['achievements']:
        user_data['achievements'].append('big_winner')
        new_achievements.append('💰 Выигрыш 10000+')

    if user_data['vip_level'] >= 3 and 'vip_player' not in
user_data['achievements']:
        user_data['achievements'].append('vip_player')
        new_achievements.append('👑 VIP игрок')

    if new_achievements:
        user_manager.update_user(user_id, user_data)
        for achievement in new_achievements:
            bot.send_message(user_id, f"🎊 Новое достижение получено:
{achievement}")


# Запуск бота
if __name__ == '__main__':
    print("🎰 BlackJack Bot запущен!")
    print(f"🃏 Инициализировано {deck_manager.deck_count} колод")
    bot.infinity_polling()
```

Результат выполнения: