Отчет по лабораторной работе №3.3

Тема: Разработка простой аркадной игры [Blackjack Classic]

A. Титульная страница

- ФИО студентов: Ефремов Николай, Зиновьев Михаил
- Студенческий билет: 245111, 245121
- Лабораторная работа №3.3: «Простые игры»
- Название игры: Blackjack Classic
- Роль в проекте:
    - Студент 1: игровой дизайн, логика игры, реализация кода
    - Студент 2: интерфейс, визуализация, документация

B. Описание игры

B.1. Назначенное задание

В рамках лабораторной работы №3.3 было выдано задание реализовать простую аркадную игру в соответствии с классическими правилами. Моим/нашим вариантом игры стал [Blackjack / Блек Джек].

Основная цель работы — применить навыки работы с двумерной анимацией и ООП при разработке простой игровой логики, а также получить опыт командной разработки, проектирования и сопровождения проекта.

B.2. Описание классической игры и её правил

[Блек Джек] — карточная игра между игроком и дилером. Цель игрока — набрать сумму очков, максимально близкую к 21, но не превышающую 21, и при этом превзойти сумму очков дилера.

Основные правила:

- Игра ведётся одной или несколькими колодами по 52 карты.
- Ценность карт:
    - 2–10 — по номиналу,
    - J, Q, K — по 10 очков,
    - A — 1 или 11 очков (в зависимости от выгоды игрока).
- В начале раунда:
    - игрок и дилер получают по две карты;
    - одна карта дилера может быть закрытой.
- Действия игрока:
    - Hit — взять ещё карту;

    - Stand — остановиться, завершить набор карт.
- После хода игрока дилер добирает карты до значения не менее 17.
- Победа:
    - если у игрока 21 с двух карт — это [Blackjack], повышенная выплата;
    - если у игрока сумма больше, чем у дилера, и ≤21 — игрок выигрывает;
    - если сумма у игрока >21 — перебор (проигрыш);
    - если суммы равны — ничья (push), ставка возвращается.

B.3. Реализованные изменения и улучшения

По сравнению с базовыми правилами реализованы:

1. Разные уровни сложности:
    - Easy: 1 колода, стартовый баланс 1500$, максимально возможная ставка 1000$.
    - Medium: 4 колоды, стартовый баланс 1000$, max bet 500$.
    - Hard: 6 колод, стартовый баланс 500$, max bet 500$.
2. Система ставок фиксированными фишками:
    - Кнопки для ставок: 10, 25, 50, 100, 500, 1000 долларов.
    - Кнопка 1000$ становится доступной только при достаточном балансе.
3. Визуальный индикатор колоды:
    - В правом верхнем углу отображается количество оставшихся карт: DECK + N cards.
    - Показано влияние параметра decks (1/4/6 колод).
4. Статистика игрока:
    - Общее количество игр;
    - Количество побед;
    - Количество поражений;
    - Количество случаев Blackjack;
    - Максимальный достигнутый баланс.

B.4. Используемые инструменты и технологии

- Язык программирования: Python 3.x
- Библиотека для 2D-графики и событий: pygame
- Формат конфигурации: JSON
- Подход к архитектуре: объектно-ориентированное программирование (ООП)
- Система управления версиями: Git (рекомендуется по ТЗ, использовалась локально).

C. Распределение ролей и задач

(адаптируйте под вашу реальную команду)

C.1. Роли участников

- Студент 1:
    - Проектирование классов Card, Deck, Player, Dealer, GameManager
    - Реализация игровой логики, подсчёт очков, обработка ставок
    - Конфигурация уровней сложности и параметров игры
- Студент 2:
    - Проектирование интерфейса и классов Renderer, Button, Menu
    - Разработка визуальной части (стол, карты, кнопки, сообщения)
    - Подготовка отчёта, диаграммы классов, скриншотов.

C.2. Методы сотрудничества и коммуникации

- Совместная работа через репозиторий Git (ветки для отдельных задач).
- Обсуждение архитектуры и распределения задач на семинарах и онлайн.
- Разделение задач по модулям (игровая логика и UI/рендер).

C.3. Распределение задач по графике

- Базовая графика реализована средствами pygame:
    - Карты — прямоугольники с текстом ранга и масти.
    - Стол — эллипс на зелёном фоне.
    - Кнопки — прямоугольники с текстом, изменяющие цвет при наведении.

D. Архитектура проекта

D.1. Структура файлов проекта

text

```
blackjack_game/
└── src/
    ├── main.py
    ├── config/
    |   ├── game_config.json
    |   └── config_loader.py
    ├── game/
    |   ├── card.py
```

```
│   ├── deck.py
│   ├── player.py
│   ├── renderer.py
│   └── game_manager.py
└── ui/
    ├── button.py
    └── menu.py
```

D.2. Диаграмма классов (описание для class_diagram.png)

Основные классы и связи:

- ConfigLoader
    - Загружает и сохраняет настройки из game_config.json.
- Card
    - Атрибуты: suit, rank, value, suit_symbol, face_up.
- Deck
    - Содержит список Card.
    - Операции: create_deck(), shuffle(), deal_card(), cards_remaining().
- Player
    - Атрибуты: name, balance, hand, bet, флаги состояний (is_busted, has_blackjack).
    - Методы: add_card(), get_hand_value(), place_bet(), win(), push(), reset_hand().
- Dealer (наследует Player)
    - Дополнительно: stand_value, should_hit(), hide_first_card(), reveal_cards().
- Button
    - Отвечает за отображение и взаимодействие с кнопкой (рисование, hover, клик).
- Renderer
    - Отвечает только за отрисовку: фона, карт, рук, текста, сообщений, индикатора колоды.
- GameManager
    - Управляет процессом игры: раздача, ходы игрока и дилера, определение результата, работа с ставками и статистикой.
- Menu
    - Управляет экранами: главное меню, настройки, статистика.
- BlackjackGame (в main.py)

- Точка входа, главный игровой цикл, создание всех объектов и переключение между меню и игрой.

D.3. Использованные шаблоны и подходы

- Инкапсуляция:
  - Каждая подсистема спрятана в своем классе (логика, UI, конфигурация, отрисовка).
- Наследование:
  - Dealer наследует логику Player и расширяет её поведением дилера.
- Композиция:
  - GameManager использует Deck, Player, Dealer, Renderer.
  - Menu использует Button и Renderer.
- Разделение ответственности (SRP):
  - Логика игры (GameManager) отделена от отрисовки (Renderer) и от конфигурации (ConfigLoader).

E. Реализованный функционал

E.1. Основные требования

1. Классический вариант игры:
   - Раздача двух карт игроку и дилеру.
   - Ходы игрока (Hit/Stand).
   - Ход дилера по правилам (до 17 очков).
   - Подсчёт очков с тузами (1 или 11).
2. Система подсчёта очков:
   - Тузы пересчитываются с 11 на 1 при переборе.
3. Управление с клавиатуры/мыши:
   - Основное управление мышью (кнопки).
4. Базовый игровой интерфейс:
   - Игровой стол, карты, кнопки действий и ставок.
5. Система состояний игры:
   - betting (ставка) → playing (ход игрока) → dealer_turn → round_over → новый раунд.

E.2. Расширенный функционал

- Уровни сложности (Easy/Medium/Hard).
- Индикатор количества оставшихся карт в колоде.
- Статистика игрока и отображение в меню.

- Крупная ставка $1000 на лёгком уровне сложности.

E.3. Скриншоты (описание)

Рекомендуемые скриншоты для вложения в отчёт:

1. Главное меню (Play/Settings/Stats/Exit).
2. Экран ставок с кнопками $10–$1000.
3. Игровой процесс (карты игрока и дилера, кнопки Hit/Stand).
4. Экран с индикатором колоды (DECK: N cards).
5. Экран статистики.

E.4. Ключевые фрагменты кода

1. Подсчёт значения руки с тузами (класс Player, get_hand_value).
2. Логика дилера (метод Dealer.should_hit).
3. Определение победителя (метод GameManager.determine_winner).
4. Индикатор колоды (метод Renderer.draw_deck_info).

F. Инструкции по запуску и игре

F.1. Запуск проекта

1. Установить Python 3.x.
2. Установить зависимости:

bash

pip install pygame==2.5.2

3. Запуск:

bash

cd blackjack_game/src

python main.py

F.2. Схема управления

- Мышь:
    - Клик по кнопкам меню: выбор режима.
    - Клик по кнопкам ставок: выбор размера ставки.
    - Клик по HIT: взять карту.
    - Клик по STAND: завершить ход.
    - Клик по NEW ROUND: начать новый раунд.
    - Клик по MENU: вернуться в главное меню.

F.3. Правила и цели игры

- Цель: набрать сумму очков ближе к 21, чем дилер, не превысив 21.
- Победа — увеличивает баланс, поражение — уменьшает.
- Blackjack (21 с двух карт) даёт повышенную выплату.

F.4. Системные требования

- ОС: Windows / Linux / macOS
- Python 3.x
- Установленный pygame
- ОЗУ: достаточно 2 ГБ+
- Видеоресурсы: интегрированной графики достаточно.

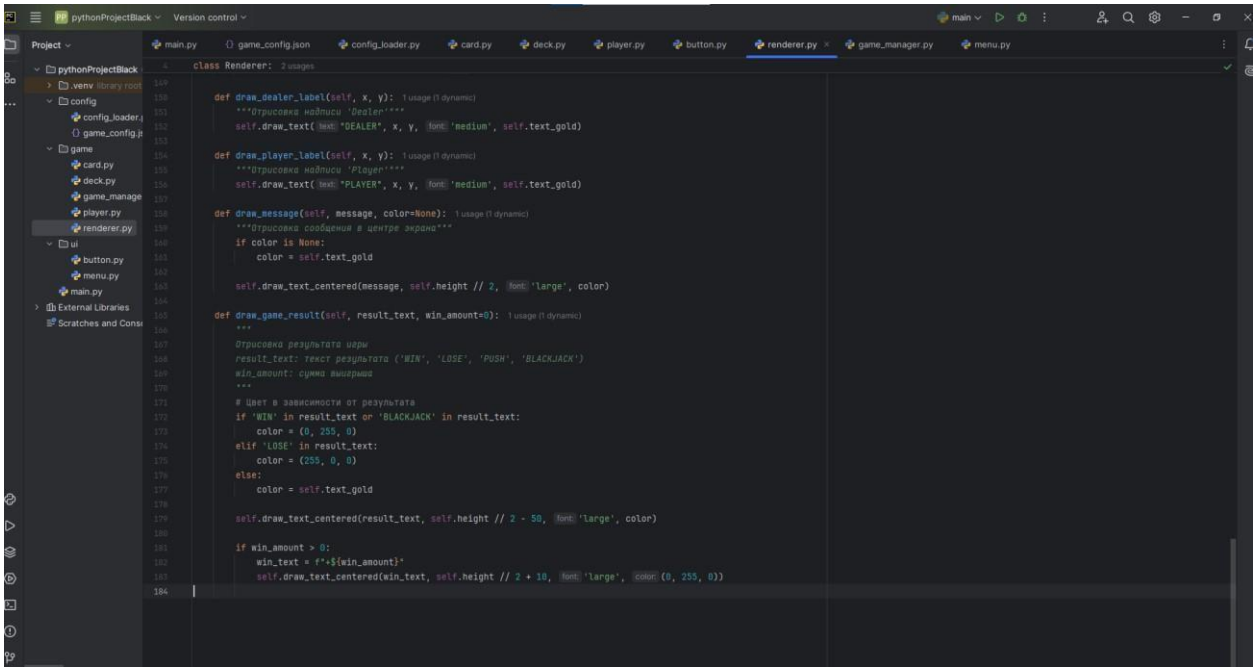
G. Полный исходный код

В состав проекта входят:

- main.py — точка входа, главный цикл.
- config_loader.py, game_config.json — конфигурация.
- card.py, deck.py, player.py, dealer (наследование), game_manager.py — игровая логика.
- renderer.py — отрисовка.
- button.py, menu.py — интерфейс и взаимодействие с пользователем.

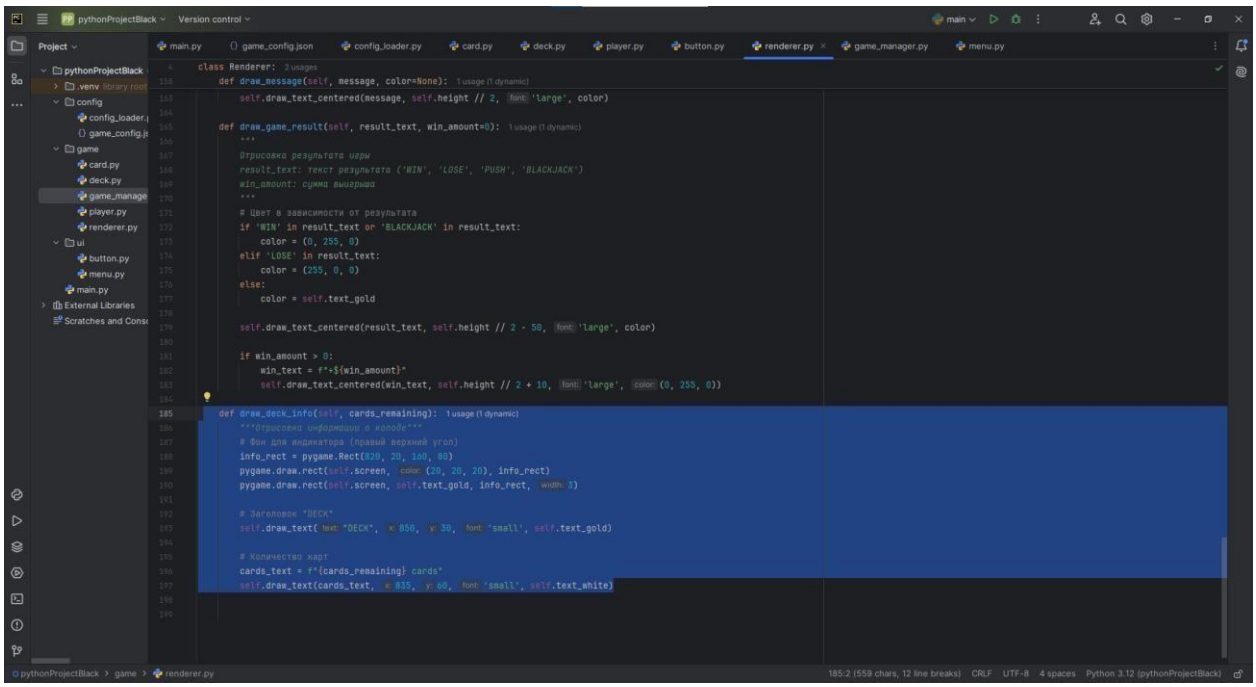
Дополнительны задания после сдачи:

до (renderer.py):



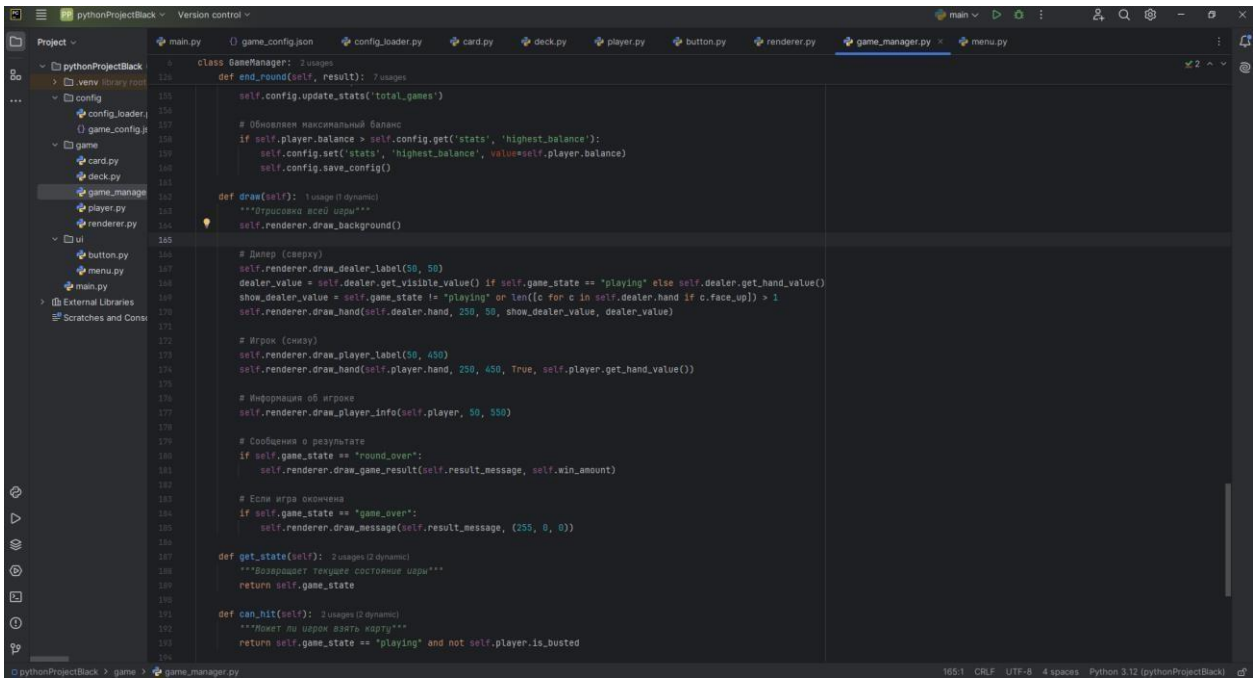
```
class Renderer:  2 usages

    def draw_dealer_label(self, x, y):  1 usage (1 dynamic)
        """Отрисовка надписи 'Dealer'"""
        self.draw_text( text: "DEALER", x, y,  font: 'medium', self.text_gold)

    def draw_player_label(self, x, y):  1 usage (1 dynamic)
        """Отрисовка надписи 'Player'"""
        self.draw_text( text: "PLAYER", x, y,  font: 'medium', self.text_gold)

    def draw_message(self, message, color=None):  1 usage (1 dynamic)
        """Отрисовка сообщения в центре экрана"""
        if color is None:
            color = self.text_gold

        self.draw_text_centered(message, self.height // 2,  font: 'large', color)

    def draw_game_result(self, result_text, win_amount=0):  1 usage (1 dynamic)
        """
        Отрисовка результата игры
        result_text: текст результата ('WIN', 'LOSE', 'PUSH', 'BLACKJACK')
        win_amount: сумма выигрыша
        """
        # Цвет в зависимости от результата
        if 'WIN' in result_text or 'BLACKJACK' in result_text:
            color = (0, 255, 0)
        elif 'LOSE' in result_text:
            color = (255, 0, 0)
        else:
            color = self.text_gold

        self.draw_text_centered(result_text, self.height // 2 - 50,  font: 'large', color)

        if win_amount > 0:
            win_text = f"+${win_amount}"
            self.draw_text_centered(win_text, self.height // 2 + 10,  font: 'large',  color: (0, 255, 0))
```

после (render.py):

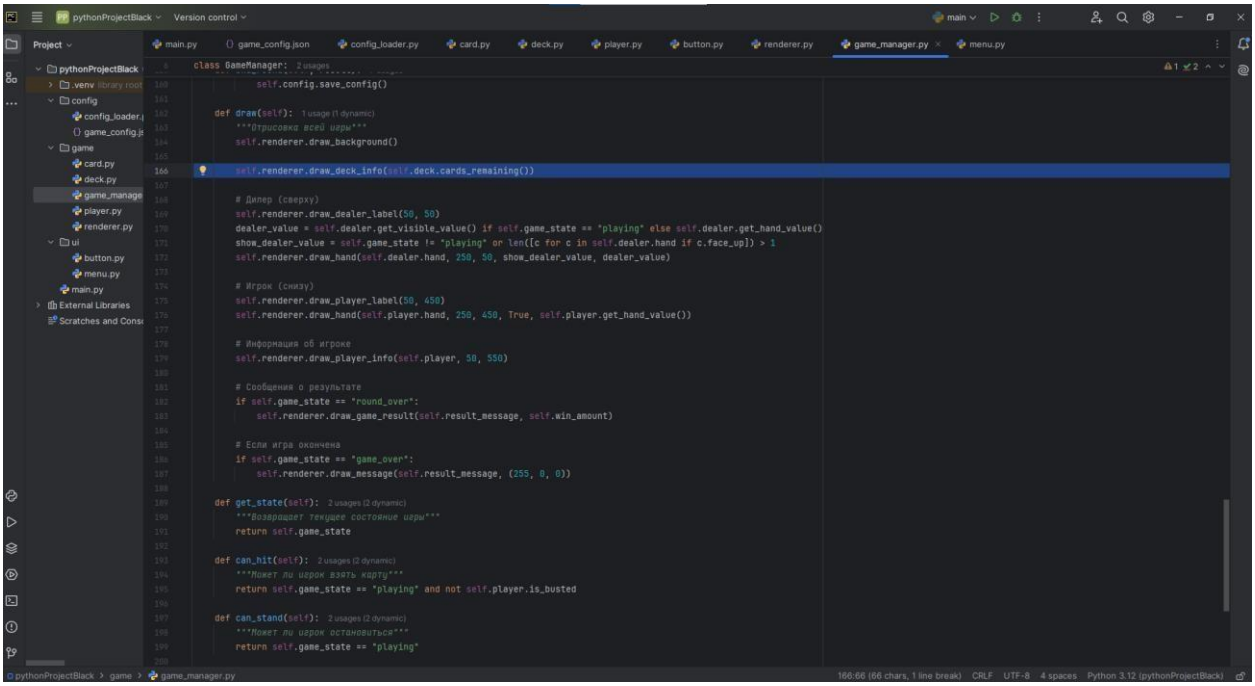


```
class Renderer:  2 usages
    def draw_message(self, message, color=None):  1 usage (1 dynamic)
        self.draw_text_centered(message, self.height // 2,  font: 'large', color)

    def draw_game_result(self, result_text, win_amount=0):  1 usage (1 dynamic)
        """
        Отрисовка результата игры
        result_text: текст результата ('WIN', 'LOSE', 'PUSH', 'BLACKJACK')
        win_amount: сумма выигрыша
        """
        # Цвет в зависимости от результата
        if 'WIN' in result_text or 'BLACKJACK' in result_text:
            color = (0, 255, 0)
        elif 'LOSE' in result_text:
            color = (255, 0, 0)
        else:
            color = self.text_gold

        self.draw_text_centered(result_text, self.height // 2 - 50,  font: 'large', color)

        if win_amount > 0:
            win_text = f"+${win_amount}"
            self.draw_text_centered(win_text, self.height // 2 + 10,  font: 'large',  color: (0, 255, 0))

    def draw_deck_info(self, cards_remaining):  1 usage (1 dynamic)
        """Отрисовка информации о колоде"""
        # Фон для индикатора (правый верхний угол)
        info_rect = pygame.Rect(820, 20, 180, 80)
        pygame.draw.rect(self.screen,  color: (20, 20, 20), info_rect)
        pygame.draw.rect(self.screen, self.text_gold, info_rect,  width: 3)

        # Заголовок "DECK"
        self.draw_text( text: "DECK",  x: 850,  y: 30,  font: 'small', self.text_gold)

        # Количество карт
        cards_text = f"{cards_remaining} cards"
        self.draw_text(cards_text,  x: 835,  y: 60,  font: 'small', self.text_white)
```

до (game_manager.py):

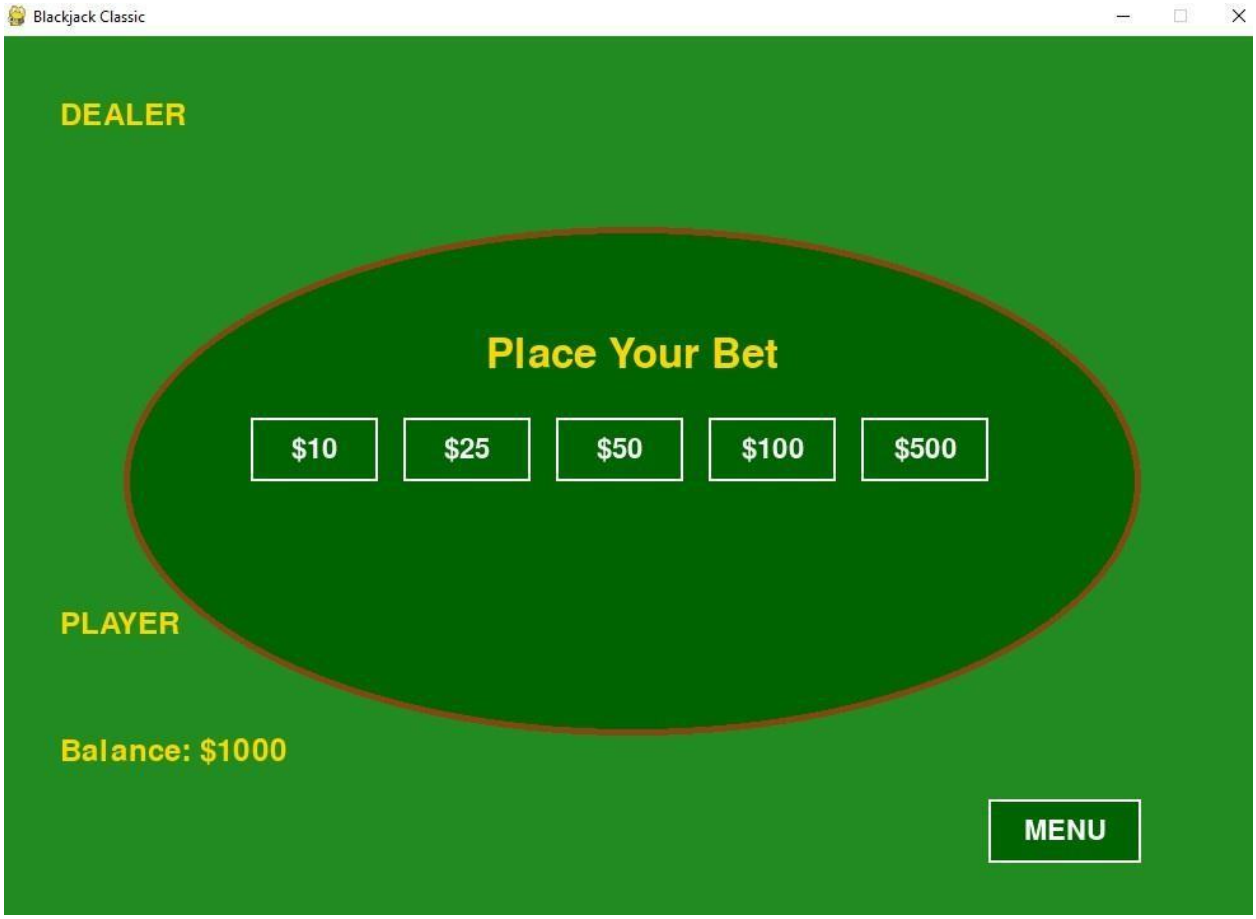

pythonProjectBlack
Version control
main
Project
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
pythonProjectBlack
.venv library root
config
config_loader.
game_config.js
game
card.py
deck.py
game_manage
player.py
renderer.py
ui
button.py
menu.py
main.py
External Libraries
Scratches and Cons
class GameManager: 2 usages
def end_round(self, result): 7 usages
self.config.update_stats('total_games')
# Обновляем максимальный баланс
if self.player.balance > self.config.get('stats', 'highest_balance'):
self.config.set('stats', 'highest_balance', value=self.player.balance)
self.config.save_config()
def draw(self): 1 usage (1 dynamic)
"""Отрисовка всей игры"""
self.renderer.draw_background()
# Дилер (сверху)
self.renderer.draw_dealer_label(50, 50)
dealer_value = self.dealer.get_visible_value() if self.game_state == "playing" else self.dealer.get_hand_value()
show_dealer_value = self.game_state != "playing" or len([c for c in self.dealer.hand if c.face_up]) > 1
self.renderer.draw_hand(self.dealer.hand, 250, 50, show_dealer_value, dealer_value)
# Игрок (снизу)
self.renderer.draw_player_label(50, 450)
self.renderer.draw_hand(self.player.hand, 250, 450, True, self.player.get_hand_value())
# Информация об игроке
self.renderer.draw_player_info(self.player, 50, 550)
# Сообщения о результате
if self.game_state == "round_over":
self.renderer.draw_game_result(self.result_message, self.win_amount)
# Если игра окончена
if self.game_state == "game_over":
self.renderer.draw_message(self.result_message, (255, 0, 0))
def get_state(self): 2 usages (2 dynamic)
"""Возвращает текущее состояние игры"""
return self.game_state
def can_hit(self): 2 usages (2 dynamic)
"""Может ли игрок взять карту"""
return self.game_state == "playing" and not self.player.is_busted
pythonProjectBlack > game > game_manager.py
165:1 CRLF UTF-8 4 spaces Python 3.12 (pythonProjectBlack)

после (game_manager.py):

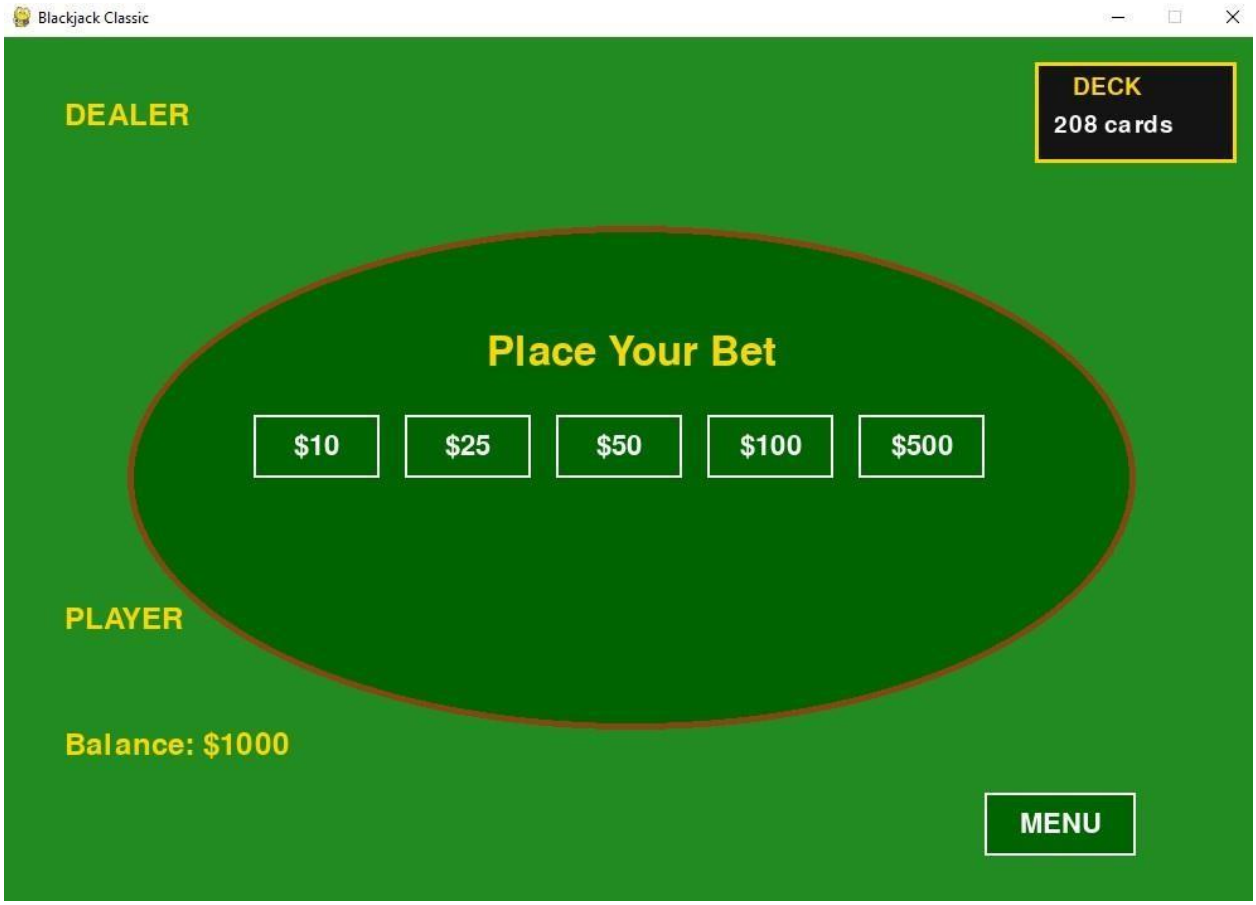


```
class GameManager:  2 usages
            self.config.save_config()

    def draw(self):  1 usage (1 dynamic)
        """Отрисовка всей игры"""
        self.renderer.draw_background()

        self.renderer.draw_deck_info(self.deck.cards_remaining())

        # Дилер (сверху)
        self.renderer.draw_dealer_label(50, 50)
        dealer_value = self.dealer.get_visible_value() if self.game_state == "playing" else self.dealer.get_hand_value()
        show_dealer_value = self.game_state != "playing" or len([c for c in self.dealer.hand if c.face_up]) > 1
        self.renderer.draw_hand(self.dealer.hand, 250, 50, show_dealer_value, dealer_value)

        # Игрок (снизу)
        self.renderer.draw_player_label(50, 450)
        self.renderer.draw_hand(self.player.hand, 250, 450, True, self.player.get_hand_value())

        # Информация об игроке
        self.renderer.draw_player_info(self.player, 50, 550)

        # Сообщения о результате
        if self.game_state == "round_over":
            self.renderer.draw_game_result(self.result_message, self.win_amount)

        # Если игра окончена
        if self.game_state == "game_over":
            self.renderer.draw_message(self.result_message, (255, 0, 0))

    def get_state(self):  2 usages (2 dynamic)
        """Возвращает текущее состояние игры"""
        return self.game_state

    def can_hit(self):  2 usages (2 dynamic)
        """Может ли игрок взять карту"""
        return self.game_state == "playing" and not self.player.is_busted

    def can_stand(self):  2 usages (2 dynamic)
        """Может ли игрок остановиться"""
        return self.game_state == "playing"
```

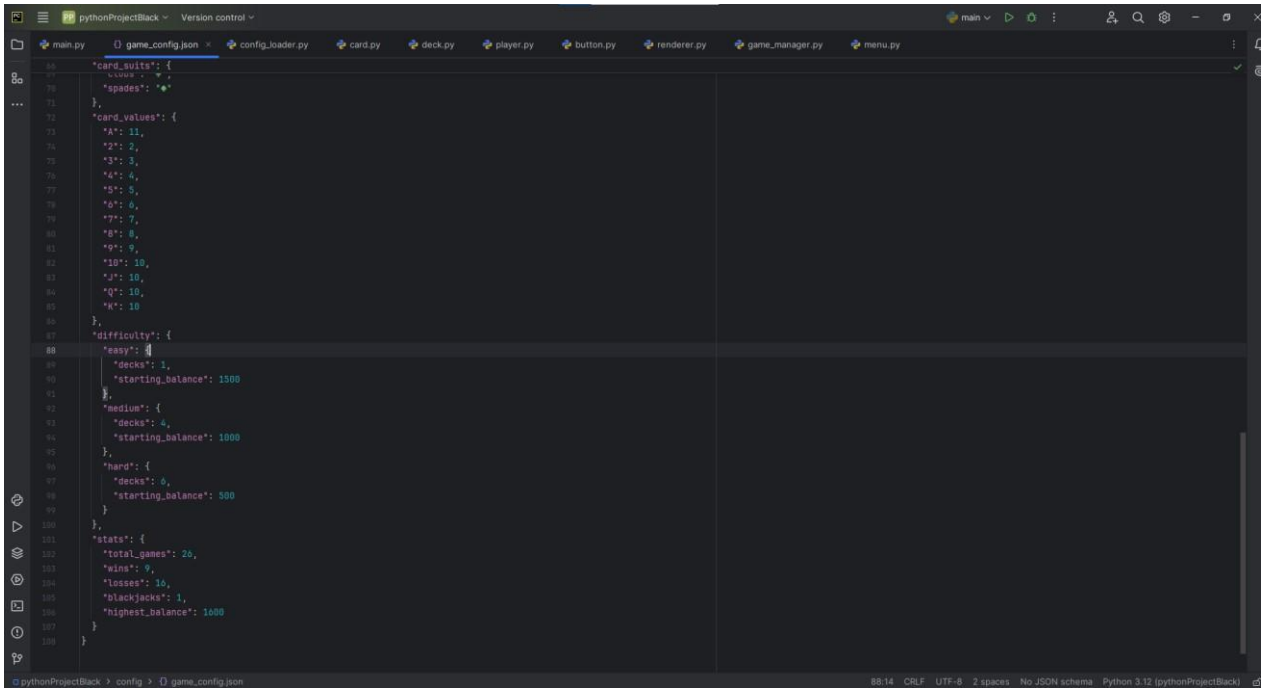
игра до модификации:



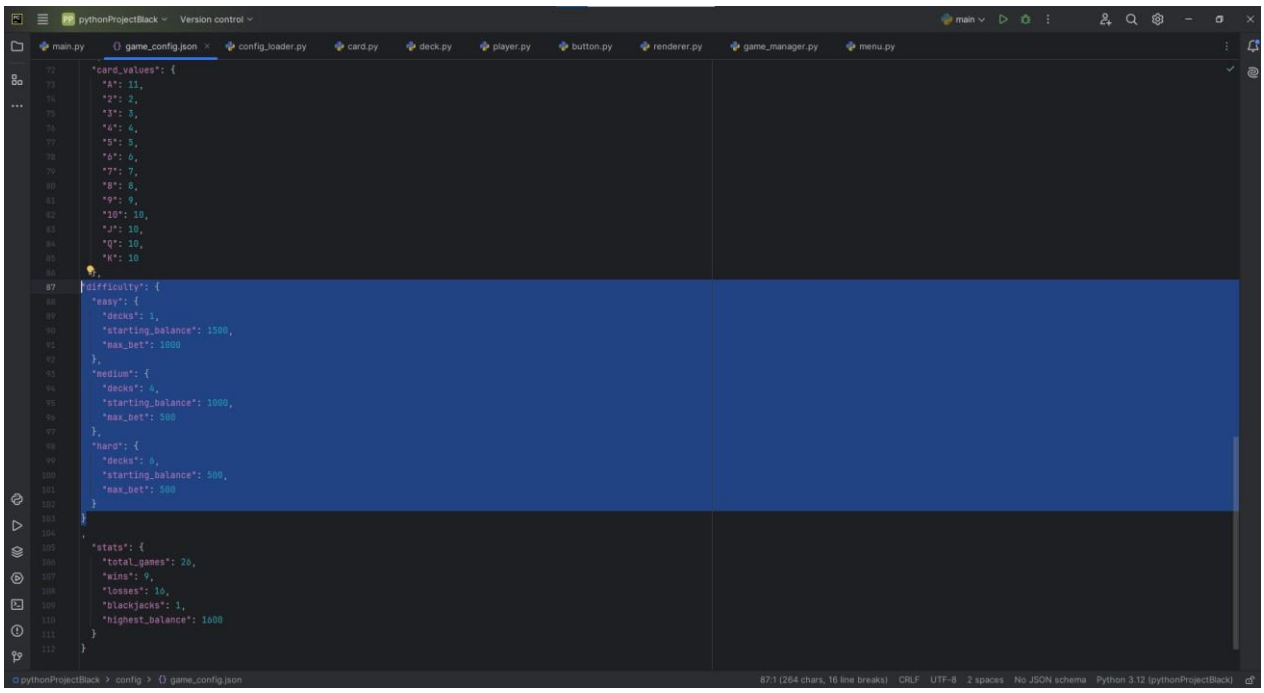
игра после модификации:



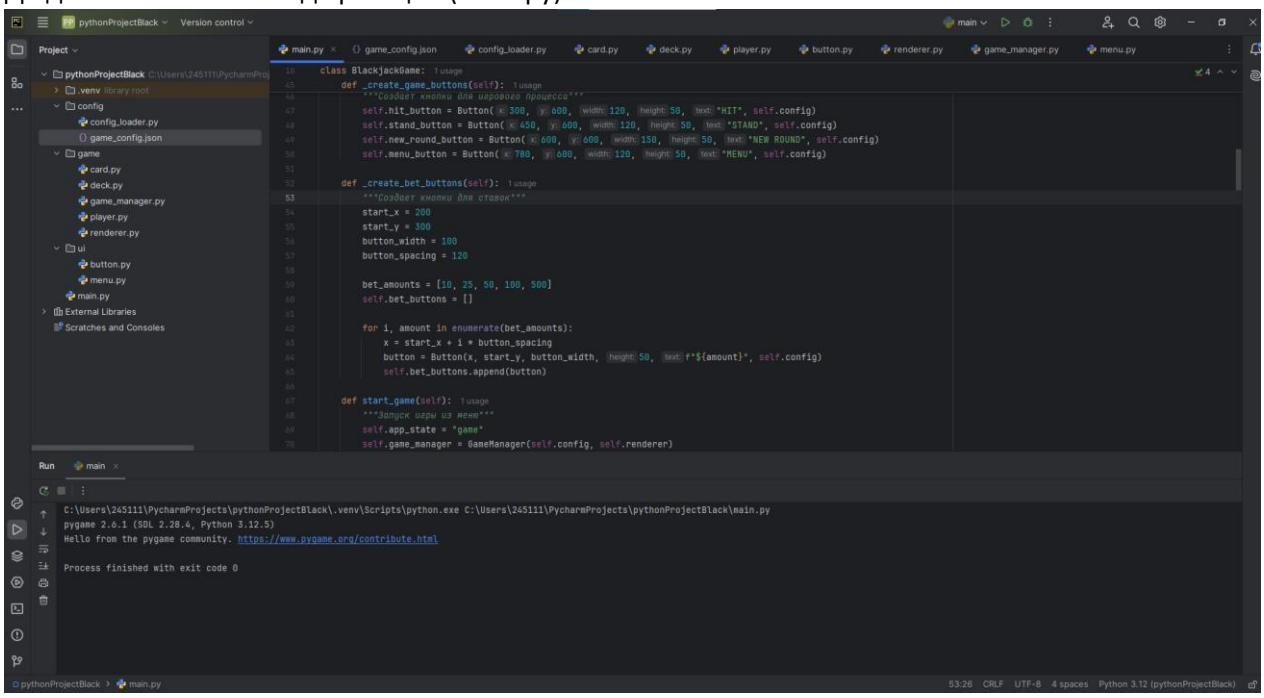
До 2-ой модификации (game_config.json):



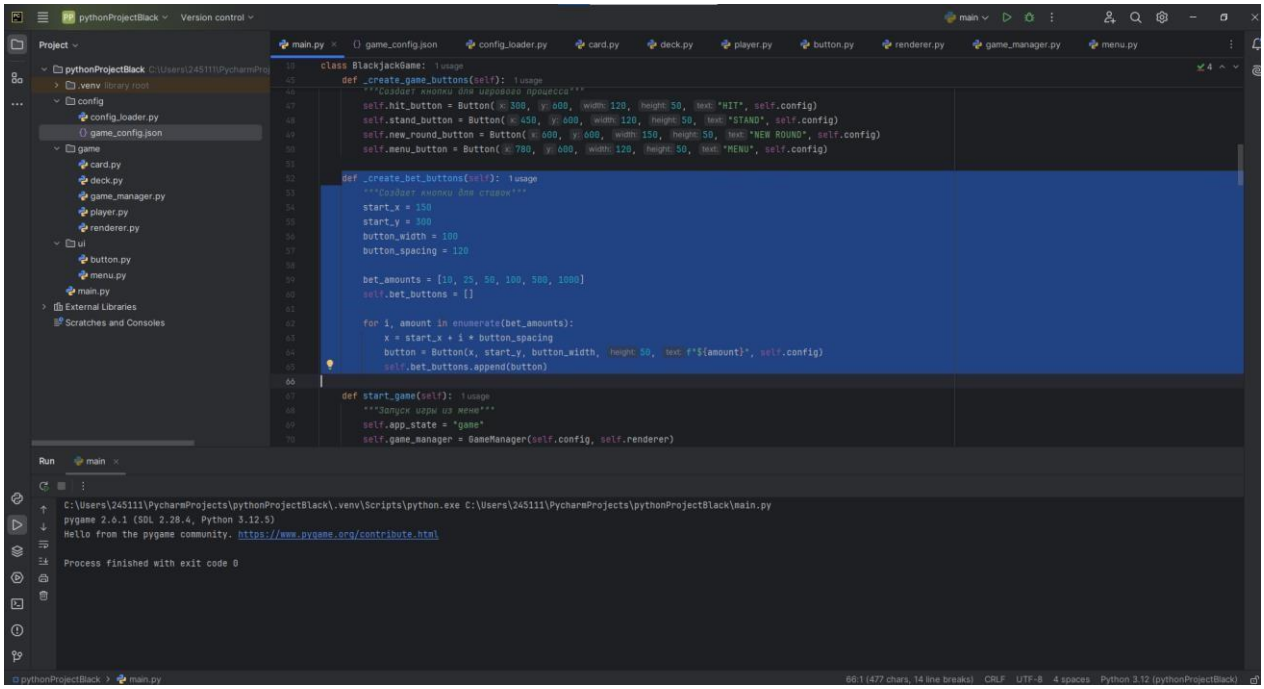
После добавления 2-ой модификации:



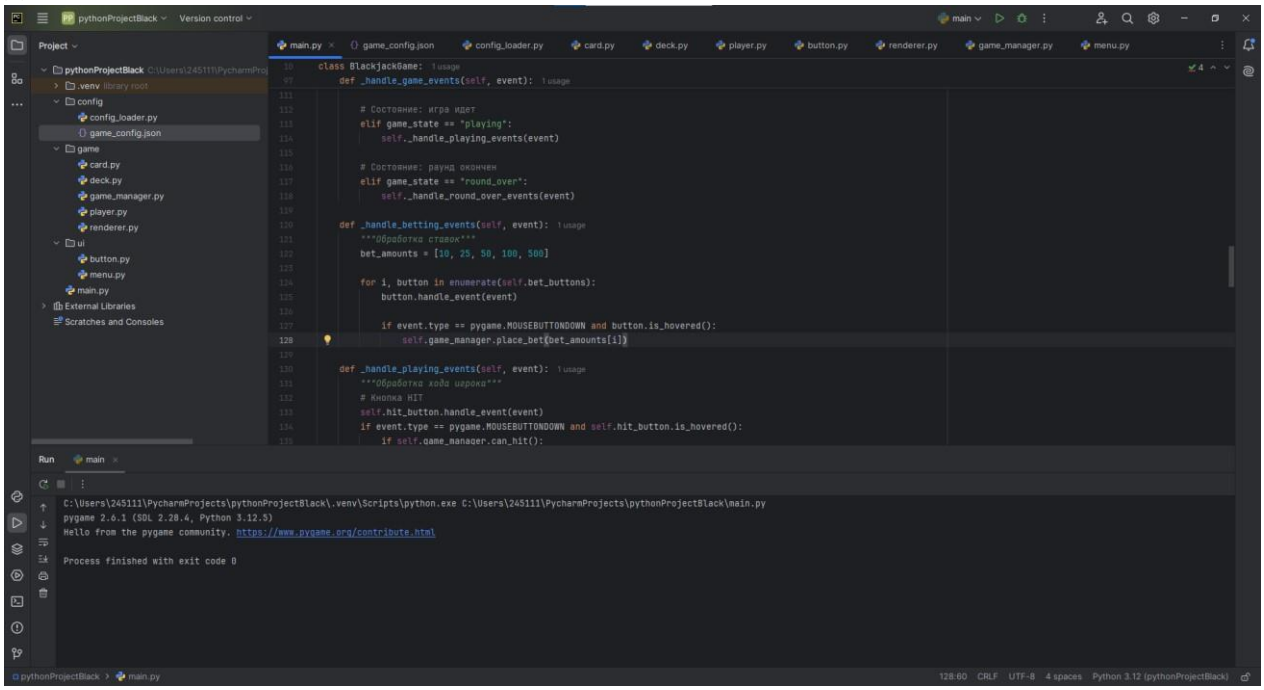
До добавления 2-ой модификации (main.py):



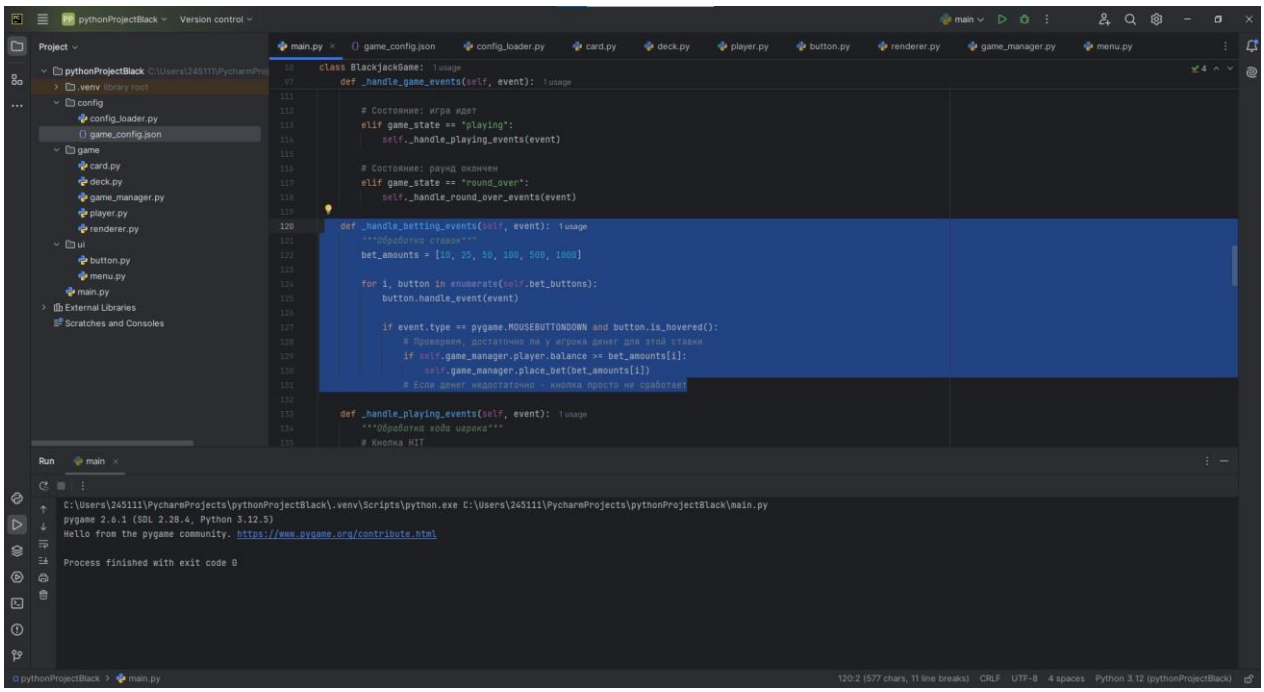
После добавления 2-ой модификации:


pythonProjectBlack
Version control
main
Project
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
class BlackjackGame:
def _create_game_buttons(self):
"""Создает кнопки для игрового процесса"""
self.hit_button = Button(x: 300, y: 600, width: 120, height: 50, text: "HIT", self.config)
self.stand_button = Button(x: 430, y: 600, width: 120, height: 50, text: "STAND", self.config)
self.new_round_button = Button(x: 600, y: 600, width: 150, height: 50, text: "NEW ROUND", self.config)
self.menu_button = Button(x: 780, y: 600, width: 120, height: 50, text: "MENU", self.config)
def _create_bet_buttons(self):
"""Создает кнопки для ставок"""
start_x = 150
start_y = 300
button_width = 100
button_spacing = 120
bet_amounts = [10, 25, 50, 100, 500, 1000]
self.bet_buttons = []
for i, amount in enumerate(bet_amounts):
x = start_x + i * button_spacing
button = Button(x, start_y, button_width, height: 50, text: f"${amount}", self.config)
self.bet_buttons.append(button)
def start_game(self):
"""Запуск игры из меню"""
self.app_state = "game"
self.game_manager = GameManager(self.config, self.renderer)
Run
main
C:\Users\245111\PycharmProjects\pythonProjectBlack\.venv\Scripts\python.exe C:\Users\245111\PycharmProjects\pythonProjectBlack\main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Process finished with exit code 0

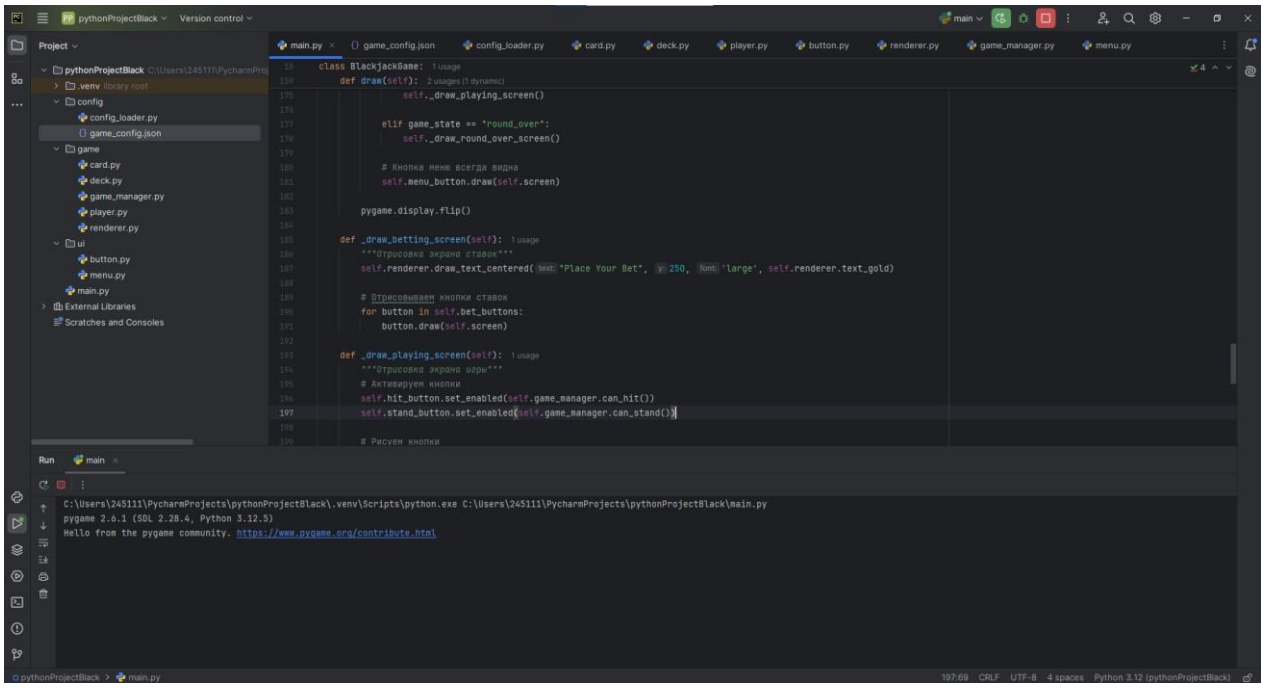
До:


pythonProjectBlack
Version control
main
Project
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
class BlackjackGame:
def _handle_game_events(self, event):
# Состояние: игра идет
elif game_state == "playing":
self._handle_playing_events(event)
# Состояние: раунд окончен
elif game_state == "round_over":
self._handle_round_over_events(event)
def _handle_betting_events(self, event):
"""Обработка ставок"""
bet_amounts = [10, 25, 50, 100, 500]
for i, button in enumerate(self.bet_buttons):
button.handle_event(event)
if event.type == pygame.MOUSEBUTTONDOWN and button.is_hovered():
self.game_manager.place_bet(bet_amounts[i])
def _handle_playing_events(self, event):
"""Обработка хода игрока"""
# Кнопка HIT
self.hit_button.handle_event(event)
if event.type == pygame.MOUSEBUTTONDOWN and self.hit_button.is_hovered():
if self.game_manager.can_hit():
Run
main
C:\Users\245111\PycharmProjects\pythonProjectBlack\.venv\Scripts\python.exe C:\Users\245111\PycharmProjects\pythonProjectBlack\main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Process finished with exit code 0

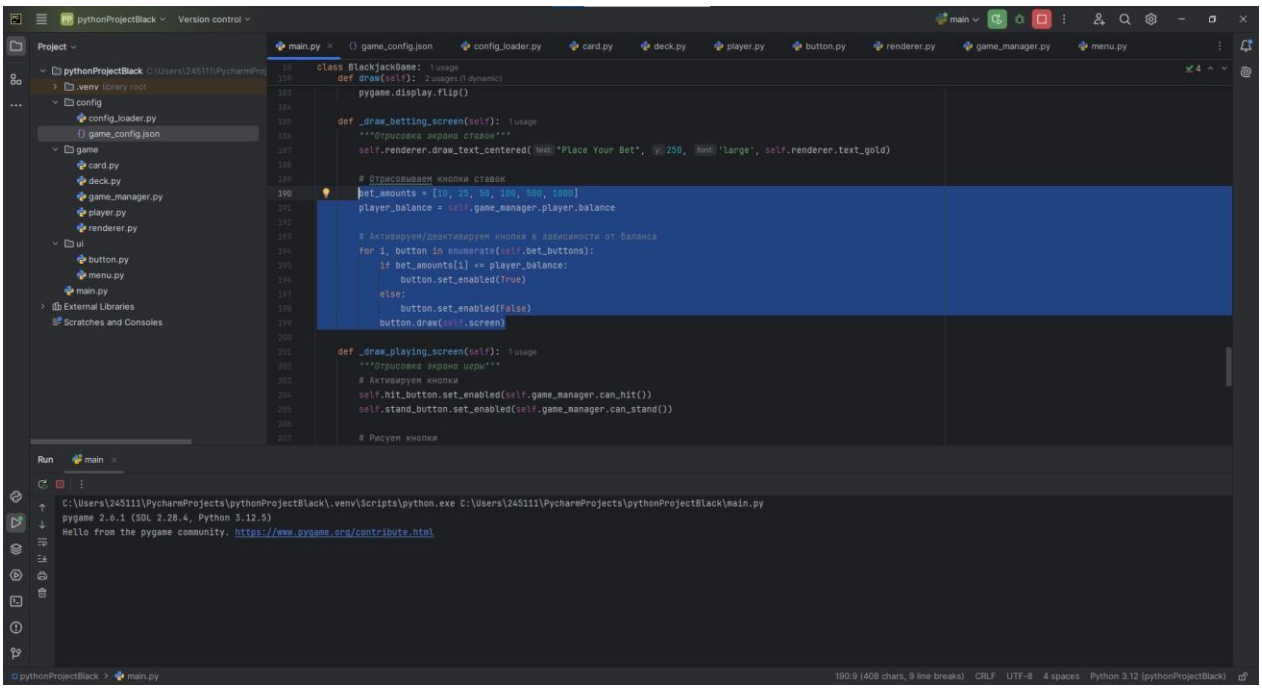
После:


pythonProjectBlack
Version control
main
Project
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
class BlackjackGame:
def _handle_game_events(self, event):
# Состояние: игра идет
elif game_state == "playing":
self._handle_playing_events(event)
# Состояние: раунд окончен
elif game_state == "round_over":
self._handle_round_over_events(event)
def _handle_betting_events(self, event):
"""Обработка ставок"""
bet_amounts = [10, 25, 50, 100, 500, 1000]
for i, button in enumerate(self.bet_buttons):
button.handle_event(event)
if event.type == pygame.MOUSEBUTTONDOWN and button.is_hovered():
# Проверяем, достаточно ли у игрока денег для этой ставки
if self.game_manager.player.balance >= bet_amounts[i]:
self.game_manager.place_bet(bet_amounts[i])
# Если денег недостаточно - кнопка просто не сработает
def _handle_playing_events(self, event):
"""Обработка хода игрока"""
Кнопка HIT
Run
main
C:\Users\245111\PycharmProjects\pythonProjectBlack\.venv\Scripts\python.exe C:\Users\245111\PycharmProjects\pythonProjectBlack\main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Process finished with exit code 0

До:


pythonProjectBlack
Version control
main
Project
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
class BlackjackGame:
def draw(self):
self._draw_playing_screen()
elif game_state == "round_over":
self._draw_round_over_screen()
# Кнопка меню всегда видна
self.menu_button.draw(self.screen)
pygame.display.flip()
def _draw_betting_screen(self):
"""Отрисовка экрана ставок"""
self.renderer.draw_text_centered(text: "Place Your Bet", y: 250, font: 'large', self.renderer.text_gold)
# Отрисовываем кнопки ставок
for button in self.bet_buttons:
button.draw(self.screen)
def _draw_playing_screen(self):
"""Отрисовка экрана игры"""
# Активируем кнопки
self.hit_button.set_enabled(self.game_manager.can_hit())
self.stand_button.set_enabled(self.game_manager.can_stand())
# Рисуем кнопки
Run
main
C:\Users\245111\PycharmProjects\pythonProjectBlack\.venv\Scripts\python.exe C:\Users\245111\PycharmProjects\pythonProjectBlack\main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html

После:


pythonProjectBlack
Version control
main
Project
pythonProjectBlack C:\Users\245111\PycharmProj
.venv library root
config
config_loader.py
game_config.json
game
card.py
deck.py
game_manager.py
player.py
renderer.py
ui
button.py
menu.py
main.py
External Libraries
Scratches and Consoles
main.py
game_config.json
config_loader.py
card.py
deck.py
player.py
button.py
renderer.py
game_manager.py
menu.py
class BlackjackGame: 1 usage
def draw(self): 2 usages (1 dynamic)
pygame.display.flip()
def _draw_betting_screen(self): 1 usage
"""Отрисовка экрана ставок"""
self.renderer.draw_text_centered(text: 'Place Your Bet', y: 250, font: 'large', self.renderer.text_gold)
# Отрисовываем кнопки ставок
bet_amounts = [10, 25, 50, 100, 500, 1000]
player_balance = self.game_manager.player.balance
# Активируем/деактивируем кнопки в зависимости от баланса
for i, button in enumerate(self.bet_buttons):
if bet_amounts[i] <= player_balance:
button.set_enabled(True)
else:
button.set_enabled(False)
button.draw(self.screen)
def _draw_playing_screen(self): 1 usage
"""Отрисовка экрана игры"""
# Активируем кнопки
self.hit_button.set_enabled(self.game_manager.can_hit())
self.stand_button.set_enabled(self.game_manager.can_stand())
# Рисуем кнопки
Run
main
C:\Users\245111\PycharmProjects\pythonProjectBlack\.venv\Scripts\python.exe C:\Users\245111\PycharmProjects\pythonProjectBlack\main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
pythonProjectBlack
main.py
190:9 (408 chars, 9 line breaks)
CRLF
UTF-8
4 spaces
Python 3.12 (pythonProjectBlack)