Оглавление

[#main.py 2](#_Toc197900251)

[# game/character.py 41](#_Toc197900252)

[# game/items.py 54](#_Toc197900253)

[# game/locations.py 55](#_Toc197900254)

[# game/quests.py 56](#_Toc197900255)

[# game/utils.py 58](#_Toc197900256)

#main.py  
from rich.columns import Columns  
from rich.panel import Panel  
from rich.text import Text  
from rich.console import Console  
from rich.box import ROUNDED  
from copy import deepcopy  
from datetime import datetime  
import random  
import json  
import os  
import sys  
import time  
try:  
 import msvcrt # Для Windows  
except ImportError:  
 import select # Для Unix-систем  
#импорты из соседних файлов  
# main.py  
from game.character import Character, Human, Warrior, Mage, Mob  
from game.items import Item, Equipment, StackableItem  
from game.quests import Quest  
from game.quests import quest\_database  
from game.locations import Location, location\_database  
  
# константы  
MAX\_INVENTORY\_SIZE = 999999999 # инвентарь - пока бесконечность  
TURN\_TIME = 3 # Время на ход в секундах  
  
#Разметка цветом  
console = Console()  
# Настраиваемая Панель действий  
class ActionBindingsManager:  
 def \_\_init\_\_(self, config\_path="save\\action\_bindings.json"):  
 self.config\_path = config\_path  
 self.bindings = {  
 "1": "attack",  
 "2": "escape",  
 "3": "heal",  
 "4": "strong\_attack",  
 "5": None,  
 "6": None,  
 "7": None,  
 "8": None,  
 "9": None,  
 "0": None  
 }  
 self.available\_actions = [  
 "attack",  
 "escape",  
 "heal",  
 "strong\_attack"  
 ]  
 self.load\_config()  
  
 def load\_config(self):  
 try:  
 if os.path.exists(self.config\_path):  
 with open(self.config\_path, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
 self.bindings.update(data.get("bindings", {}))  
 self.available\_actions = data.get("available\_actions", self.available\_actions)  
 except Exception as e:  
 print(f"Error loading action bindings: {e}")  
  
 def save\_config(self):  
 try:  
 os.makedirs(os.path.dirname(self.config\_path), exist\_ok=True)  
 with open(self.config\_path, 'w', encoding='utf-8') as f:  
 json.dump({  
 "bindings": self.bindings,  
 "available\_actions": self.available\_actions  
 }, f, indent=4)  
 except Exception as e:  
 print(f"Error saving action bindings: {e}")  
  
 def get\_action(self, key):  
 return self.bindings.get(key)  
  
 def set\_binding(self, key, action):  
 if key in self.bindings and action in self.available\_actions:  
 self.bindings[key] = action  
 return True  
 return False  
  
 def get\_available\_actions(self):  
 return self.available\_actions  
  
 def add\_new\_action(self, action\_name):  
 if action\_name not in self.available\_actions:  
 self.available\_actions.append(action\_name)  
 return True  
 return False  
action\_manager = ActionBindingsManager() #Инициализация в начале игры  
  
  
def configure\_action\_panel(action\_manager):  
 *"""Меню настройки панели действий"""* while True:  
 clear\_screen()  
  
 # Показываем текущие настройки  
 bindings\_text = Text()  
 bindings\_text.append("Текущие настройки панели действий:\n", style="bold underline")  
  
 for key in sorted(action\_manager.bindings.keys()):  
 action = action\_manager.bindings[key]  
 bindings\_text.append(f"{key}: {action if action else 'Не назначено'}\n")  
  
 console.print(Panel(bindings\_text, title="Настройки панели действий", border\_style="blue"))  
  
 # Показываем доступные действия  
 actions\_text = Text()  
 actions\_text.append("Доступные действия:\n", style="bold underline")  
 for i, action in enumerate(action\_manager.get\_available\_actions(), 1):  
 actions\_text.append(f"{i}. {action}\n")  
  
 console.print(Panel(actions\_text, title="Доступные действия", border\_style="green"))  
  
 # Меню управления  
 menu\_text = Text()  
 menu\_text.append("Команды:\n", style="bold underline")  
 menu\_text.append("[Н] - Назначить действие на клавишу\n")  
 menu\_text.append("[С] - Сбросить настройки по умолчанию\n")  
 menu\_text.append("[В] - Выйти из настроек\n")  
 console.print(Panel(menu\_text, title="Управление", border\_style="yellow"))  
  
 choice = input("Выберите действие: ").strip().lower()  
  
 if choice in ["н", "назначить"]:  
 key = input("Введите цифру (1-9, 0) для настройки: ").strip()  
 if key not in action\_manager.bindings:  
 console.print("[red]Неверная клавиша![/red]")  
 input("Нажмите Enter чтобы продолжить...")  
 continue  
  
 action\_num = input("Введите номер действия для назначения: ").strip()  
 try:  
 action\_num = int(action\_num)  
 if 1 <= action\_num <= len(action\_manager.available\_actions):  
 action = action\_manager.available\_actions[action\_num - 1]  
 action\_manager.set\_binding(key, action)  
 action\_manager.save\_config()  
 console.print(f"[green]Действие '{action}' назначено на клавишу '{key}'[/green]")  
 else:  
 console.print("[red]Неверный номер действия![/red]")  
 except ValueError:  
 console.print("[red]Введите число![/red]")  
  
 input("Нажмите Enter чтобы продолжить...")  
  
 elif choice in ["с", "сбросить"]:  
 action\_manager.bindings = {  
 "1": "attack",  
 "2": "escape",  
 "3": "heal",  
 "4": "strong\_attack",  
 "5": None,  
 "6": None,  
 "7": None,  
 "8": None,  
 "9": None,  
 "0": None  
 }  
 action\_manager.save\_config()  
 console.print("[green]Настройки сброшены к значениям по умолчанию[/green]")  
 input("Нажмите Enter чтобы продолжить...")  
  
 elif choice in ["в", "выход"]:  
 break  
  
 else:  
 console.print("[red]Неверная команда![/red]")  
 input("Нажмите Enter чтобы продолжить...")  
  
#класс панель  
class GameUI:  
 def \_\_init\_\_(self, hero):  
 self.hero = hero  
 self.layout = Layout()  
  
 # Разделяем экран на части  
 self.layout.split(  
 Layout(name="header", size=3),  
 Layout(name="main", ratio=1),  
 Layout(name="footer", size=7)  
 )  
  
 # Делим основную часть на две колонки  
 self.layout["main"].split\_row(  
 Layout(name="left", ratio=2),  
 Layout(name="right", ratio=1)  
 )  
  
 self.message\_log = []  
  
 def update\_ui(self):  
 # Обновляем заголовок  
 self.layout["header"].update(  
 Panel(f"Adventures of Heroes - {self.hero.name} (Уровень: {self.hero.level})", style="bold blue")  
 )  
  
 # Левая панель - основная информация  
 stats = (  
 f"Здоровье: {self.hero.health\_points}/{self.hero.max\_health\_points()}\n"  
 f"Атака: {self.hero.attack\_power}\n"  
 f"Защита: {self.hero.defence}\n"  
 f"Опыт: {self.hero.experience}/{self.hero.exp\_base \* 2}\n"  
 f"Деньги: {self.hero.money} монет\n"  
 f"Локация: {self.hero.location.name}\n"  
 f"Убито врагов: {self.hero.count\_kill}"  
 )  
 self.layout["left"].update(  
 Panel(stats, title="Характеристики", border\_style="green")  
 )  
  
 # Правая панель - снаряжение  
 equipment = "\n".join(  
 f"{slot}: {item.name if item else 'Пусто'}"  
 for slot, item in self.hero.equipment.items()  
 )  
 self.layout["right"].update(  
 Panel(equipment, title="Снаряжение", border\_style="blue")  
 )  
  
 # Нижняя панель - сообщения и ввод  
 messages = "\n".join(self.message\_log[-5:]) if self.message\_log else "Добро пожаловать в игру!"  
 self.layout["footer"].update(  
 Panel(messages, title="Сообщения", border\_style="yellow")  
 )  
  
 def add\_message(self, message):  
 self.message\_log.append(message)  
 if len(self.message\_log) > 100: # Ограничиваем количество сообщений  
 self.message\_log.pop(0)  
  
 def get\_input(self, prompt):  
 # Временное решение для ввода - можно улучшить с помощью curses  
 self.update\_ui()  
 console.print(Panel(prompt, style="bold"))  
 return input("> ")  
  
# все функции для баров (красявости и читательность удобство)  
  
def show\_hero\_creation\_panel(hero\_name, location\_name):  
 clear\_screen()  
 """Отображает красивую панель создания героя"""  
 PANEL\_WIDTH = 42 # Ширина панели (должна соответствовать количеству символов в рамке)  
  
 creation\_text = Text()  
 creation\_text.append("\n\n", style="bold")  
  
 # Верхняя граница  
 creation\_text.append("╔" + "═" \* (PANEL\_WIDTH - 2) + "╗\n", style="bright\_cyan")  
 creation\_text.append("║" + " " \* (PANEL\_WIDTH - 2) + "║\n", style="bright\_cyan")  
  
 # Заголовок  
 creation\_text.append("║", style="bright\_cyan")  
 creation\_text.append(" " \* ((PANEL\_WIDTH - len("НОВЫЙ ГЕРОЙ СОЗДАН!")) // 2), style="bright\_cyan")  
 creation\_text.append("НОВЫЙ ГЕРОЙ СОЗДАН!", style="bright\_cyan bold blink")  
 creation\_text.append(" " \* ((PANEL\_WIDTH - len("НОВЫЙ ГЕРОЙ СОЗДАН!")-2) // 2), style="bright\_cyan")  
 creation\_text.append("║\n", style="bright\_cyan")  
  
 creation\_text.append("║" + " " \* (PANEL\_WIDTH - 2) + "║\n", style="bright\_cyan")  
 creation\_text.append("╠" + "═" \* (PANEL\_WIDTH - 2) + "╣\n", style="bright\_cyan")  
  
 # Имя героя  
 name\_line = f"Имя: {hero\_name}"  
 creation\_text.append("║ ", style="bright\_cyan")  
 creation\_text.append(name\_line, style="bright\_yellow bold")  
 creation\_text.append(" " \* (PANEL\_WIDTH - len(name\_line) - 3) + "║\n", style="bright\_cyan")  
  
 # Класс героя  
 class\_line = "Класс: Человек (Странник)"  
 creation\_text.append("║ ", style="bright\_cyan")  
 creation\_text.append(class\_line, style="bright\_yellow")  
 creation\_text.append(" " \* (PANEL\_WIDTH - len(class\_line) - 3) + "║\n", style="bright\_cyan")  
  
 # Локация  
 location\_line = f"Стартовая локация: {location\_name}"  
 creation\_text.append("║ ", style="bright\_cyan")  
 creation\_text.append(location\_line, style="bright\_yellow")  
 creation\_text.append(" " \* (PANEL\_WIDTH - len(location\_line) - 3) + "║\n", style="bright\_cyan")  
  
 # Пустая строка  
 creation\_text.append("║" + " " \* (PANEL\_WIDTH - 2) + "║\n", style="bright\_cyan")  
  
 # Совет 1  
 advice\_line1 = "Совет: Начните с исследования Храма"  
 creation\_text.append("║ ", style="bright\_cyan")  
 creation\_text.append(advice\_line1, style="italic")  
 creation\_text.append(" " \* (PANEL\_WIDTH - len(advice\_line1) - 3) + "║\n", style="bright\_cyan")  
  
 # Совет 2  
 advice\_line2 = "и попробуйте сразиться с монстрами"  
 creation\_text.append("║ ", style="bright\_cyan")  
 creation\_text.append(advice\_line2, style="italic")  
 creation\_text.append(" " \* (PANEL\_WIDTH - len(advice\_line2) - 3) + "║\n", style="bright\_cyan")  
  
 # Нижняя граница  
 creation\_text.append("╚" + "═" \* (PANEL\_WIDTH - 2) + "╝\n", style="bright\_cyan")  
  
 console.print(creation\_text)  
 input("\nНажмите Enter чтобы начать свое приключение...")  
  
  
def show\_name\_confirmation(name):  
 *"""Отображает панель подтверждения имени"""* clear\_screen()  
  
 # Создаем текст для панели  
 panel\_content = Text()  
 name\_text = Text(f"Вы выбрали имя: {name}\n\n", style="bright\_yellow")  
 name\_text.append(name, style="bold bright\_green")  
 panel\_content.append(name\_text)  
  
 question = Text("Это имя вам нравится?\n\n", style="bright\_yellow")  
 panel\_content.append(question)  
  
 options = Text()  
 options.append("1. Да, начать игру!\n", style="green")  
 options.append("2. Нет, изменить имя", style="red")  
 panel\_content.append(options)  
  
 # Создаем панель  
 panel = Panel(  
 panel\_content,  
 border\_style="bright\_yellow",  
 width=50,  
 padding=(1, 2)  
 )  
  
 console.print(panel)  
 return input("> Ваш выбор (1/2): ").strip().lower()  
  
  
def show\_name\_input\_panel():  
 *"""Отображает красивую панель ввода имени"""* clear\_screen()  
  
 # Создаем текст для панели  
 panel\_content = Text()  
 title = Text("СОЗДАНИЕ ПЕРСОНАЖА\n", style="bold bright\_cyan")  
 title.justify = "center" # Выравнивание по центру  
 panel\_content.append(title)  
 panel\_content.append("\nДайте имя вашему герою\n", style="bright\_yellow")  
 panel\_content.append("(или введите 'выход' для отмены)", style="italic")  
  
 # Создаем панель  
 panel = Panel(  
 panel\_content,  
 border\_style="bright\_blue",  
 width=50,  
 padding=(1, 2)  
 )  
  
 console.print(panel)  
 return input("\n> Введите имя героя: ").strip()  
  
def character\_creation\_flow():  
 *"""Управляет процессом создания персонажа"""* while True:  
 # Шаг 1: Ввод имени  
 hero\_name = show\_name\_input\_panel()  
  
 if hero\_name.lower() in ["выход", "в"]:  
 console.print("[bright\_red]Создание персонажа отменено[/bright\_red]")  
 time.sleep(1)  
 return None  
  
 if not hero\_name:  
 console.print("[red]Имя не может быть пустым![/red]")  
 time.sleep(1)  
 continue  
  
 # Шаг 2: Подтверждение имени  
 choice = show\_name\_confirmation(hero\_name)  
  
 if choice in ["1", "да", "д", ""]:  
 # Создаем героя  
 hero = Human(level=1, name=hero\_name)  
 hero.active\_quests = []  
 hero.completed\_quests = []  
 hero.set\_location(Location(  
 name="Храм",  
 description="Древний храм",  
 danger\_level=4,  
 zone\_type="combat",  
 id\_loc=4  
 ))  
  
 # Показываем финальный экран  
 show\_hero\_creation\_panel(hero.name, hero.location.name)  
 return hero  
  
 elif choice in ["2", "нет", "н"]:  
 console.print("[yellow]Попробуем другое имя...[/yellow]")  
 time.sleep(1)  
 continue  
  
 else:  
 console.print("[red]Неверный выбор![/red]")  
 time.sleep(1)  
  
def show\_trade\_interface(hero, merchant):  
 *"""Отображает интерфейс торговли с панелями"""* clear\_screen()  
  
 # 1. Панель предложений торговца (левая)  
 merchant\_text = Text()  
 merchant\_text.append(f"Торговец: {merchant.name}\n", style="bold underline")  
 #merchant\_text.append(f"Ваши деньги: {hero.money} монет\n", style="gold1")  
 merchant\_text.append("\nПредлагаемые товары:\n", style="bold")  
  
 for i, item in enumerate(merchant.items, 1):  
 merchant\_text.append(f"{i}. {item.name} - {item.stock\_price} монет\n")  
  
 merchant\_panel = Panel(  
 merchant\_text,  
 title="Торговец",  
 border\_style="blue",  
 width=40  
 )  
  
 # 2. Панель инвентаря героя (правая)  
 inventory\_text = Text()  
 inventory\_text.append("Ваш инвентарь:\n", style="bold underline")  
 inventory\_text.append(f"Ваши деньги: {hero.money} монет\n", style="gold1")  
  
 if not hero.inventory:  
 inventory\_text.append("Инвентарь пуст\n", style="dim")  
 else:  
 for i, item in enumerate(hero.inventory, 1):  
 if isinstance(item, StackableItem):  
 inventory\_text.append(f"{i}. {item.name} (x{item.quantity}) - {item.stock\_price // 2} монет\n")  
 else:  
 inventory\_text.append(f"{i}. {item.name} - {item.stock\_price // 2} монет\n")  
  
 inventory\_panel = Panel(  
 inventory\_text,  
 title="Инвентарь",  
 border\_style="green",  
 width=40  
 )  
  
 # 3. Панель диалога (середина)  
 dialog\_text = Text()  
 dialog\_text.append("Диалог с торговцем:\n", style="bold underline")  
  
 # Проверяем активные квесты от этого торговца  
 merchant\_quests = [q for q in hero.active\_quests if q.giver == "Торговец"]  
 if merchant\_quests:  
 dialog\_text.append("\nАктивные квесты:\n", style="bold")  
 for quest in merchant\_quests:  
 status = "Готово к сдаче!" if hero.is\_quest\_ready\_to\_complete(  
 quest.id) else f"{quest.current\_amount}/{quest.target\_amount}"  
 color = "green" if hero.is\_quest\_ready\_to\_complete(quest.id) else None  
 dialog\_text.append(f"- {quest.name}: ")  
 dialog\_text.append(status + "\n", style=color)  
  
 dialog\_panel = Panel(  
 dialog\_text,  
 title="Диалог",  
 border\_style="yellow",  
 width=40  
 )  
  
 # 4. Панель действий (нижняя)  
 actions\_text = Text()  
 actions\_text.append("Доступные действия:\n", style="bold underline")  
 actions\_text.append("[К]упить [номер] - купить предмет у торговца\n")  
 actions\_text.append("[П]родать [номер] - продать предмет из инвентаря\n")  
 actions\_text.append("[КВ]ест - получить квест от торговца\n")  
 actions\_text.append("[С]дать [номер] - сдать квест\n")  
 actions\_text.append("[В]ыход - закончить торговлю\n")  
  
 actions\_panel = Panel(  
 actions\_text,  
 title="Действия",  
 border\_style="red",  
 width=120  
 )  
  
 # Вывод всех панелей  
 console.print(Columns([merchant\_panel, dialog\_panel, inventory\_panel], expand=False))  
 console.print(actions\_panel)  
  
  
def get\_exp\_bar(current, max\_exp, width=20):  
 *"""Создает текстовую полоску опыта"""* current = max(0, int(current))  
 max\_exp = max(1, int(max\_exp))  
 percent = min(1.0, current / max\_exp)  
  
 filled = int(percent \* width)  
 empty = width - filled  
  
 exp\_bar = (  
 f"[cyan]{'█' \* filled}[/cyan]" +  
 f"[white]{'░' \* empty}[/white] " +  
 f"{current}/{max\_exp}"  
 )  
 return exp\_bar  
  
  
def get\_health\_bar(current, max\_hp, width=20):  
 *"""Создает текстовую полоску здоровья с правильным применением цветов"""* # Защита от некорректных значений  
 current = max(0, int(current))  
 max\_hp = max(1, int(max\_hp))  
 percent = min(1.0, current / max\_hp)  
  
 # Определение цвета  
 if percent < 0.33:  
 color = "red"  
 elif percent < 0.66:  
 color = "yellow"  
 else:  
 color = "green"  
  
 # Создаем полоску здоровья  
 filled = int(percent \* width)  
 empty = width - filled  
  
 # Собираем все в одну строку без переносов  
 health\_bar = (  
 f"[{color}]" +  
 '█' \* filled +  
 "[/]" +  
 "[white]" +  
 '░' \* empty +  
 "[/] " +  
 "\n" +  
 f"{current}/{max\_hp}"  
 )  
 return health\_bar  
  
  
def show\_character\_interface(hero):  
 *"""Отображает интерфейс характеристик героя с 4 панелями"""* clear\_screen()  
  
 # 1. Панель статуса (левая)  
 status\_text = Text()  
 status\_text.append(f"Имя: {hero.name}\n", style="bold")  
 status\_text.append(f"Класс: {hero.get\_class\_hero\_rus()}\n")  
 status\_text.append(f"Уровень: {hero.level}\n", style="cyan")  
  
 # Здоровье с использованием Text.from\_markup  
 health\_line = Text("Здоровье: ")  
 health\_line.append(Text.from\_markup(get\_health\_bar(hero.health\_points, hero.max\_health\_points())))  
 status\_text.append(health\_line)  
 status\_text.append("\n\n")  
  
 # Опыт с использованием Text.from\_markup  
 exp\_line = Text("Опыт: ")  
 exp\_line.append(Text.from\_markup(get\_exp\_bar(hero.experience, hero.exp\_base \* 2)))  
 status\_text.append(exp\_line)  
 status\_text.append("\n\n")  
  
 status\_text.append(f"Атака: {hero.attack\_power}\n")  
 status\_text.append(f"Защита: {hero.defence}\n")  
 status\_text.append(f"Убито врагов: {hero.count\_kill}\n")  
 status\_text.append(f"\nДеньги: {hero.money} монет\n", style="gold1")  
 status\_text.append(f"Локация: {hero.location.name}\n", style="magenta")  
  
 status\_panel = Panel(  
 status\_text,  
 title="Статус героя",  
 border\_style="blue",  
 width=40  
 )  
  
 # 2. Панель снаряжения (середина)  
 equipment\_text = Text()  
 equipment\_text.append("Снаряжение героя:\n", style="bold underline cyan")  
  
 slot\_colors = {  
 "Голова": "bright\_cyan",  
 "Тело": "bright\_blue",  
 "Руки": "bright\_yellow",  
 "Ноги": "bright\_magenta",  
 "Оружие": "bright\_red",  
 "Плащ": "bright\_green"  
 }  
  
 # Создаем список слотов с сохранением порядка  
 slots = list(hero.equipment.items())  
  
 for index, (slot, item) in enumerate(slots, start=1):  
 # Нумерация и название слота  
 equipment\_text.append(f"{index}. {slot}: ", style=slot\_colors.get(slot, "bold"))  
  
 # Предмет или "Пусто"  
 if item:  
 equipment\_text.append(item.name, style="bright\_white")  
 if isinstance(item, Equipment):  
 # Добавляем бонусы для экипировки  
 equipment\_text.append(f" (+{item.effect\_value} {item.effect})", style="bright\_green")  
 else:  
 equipment\_text.append("Пусто", style="dim italic")  
  
 equipment\_text.append("\n") # Перенос строки  
  
 equipment\_panel = Panel(  
 equipment\_text,  
 title="[gold1]Снаряжение[/gold1]",  
 border\_style="yellow",  
 width=45,  
 padding=(1, 2)  
 )  
  
 # 3. Панель инвентаря (правая)  
 inventory\_text = Text()  
 inventory\_text.append("Инвентарь:\n", style="bold underline")  
 if not hero.inventory:  
 inventory\_text.append("Инвентарь пуст\n", style="dim")  
 else:  
 for i, item in enumerate(hero.inventory, 1):  
 if isinstance(item, StackableItem):  
 inventory\_text.append(f"{i}. {item.name} (x{item.quantity})\n")  
 else:  
 inventory\_text.append(f"{i}. {item.name}\n")  
  
 inventory\_panel = Panel(  
 inventory\_text,  
 title="Инвентарь",  
 border\_style="green",  
 width=40  
 )  
  
 # 4. Панель действий (нижняя)  
 actions\_text = Text()  
 actions\_text.append("Управление инвентарем:\n", style="bold underline")  
 actions\_text.append("1. Использовать предмет:\n", style="bold")  
 actions\_text.append(" • Введите 'п [номер]' или 'предмет [номер]'\n")  
 #actions\_text.append(" • Пример: 'п 1' - использовать первый предмет\n")  
 actions\_text.append("2. Снять экипировку:\n", style="bold")  
 actions\_text.append(" • Введите 'с [номер]' или 'снять [номер]'\n")  
 actions\_text.append(" • Номера слотов: 1-Голова, 2-Тело, 3-Руки, 4-Ноги, 5-Оружие, 6-Плащ\n")  
 actions\_text.append(" • Пример: 'с 3' - снять перчатки\n")  
 actions\_text.append("3. Выход:\n", style="bold")  
 actions\_text.append(" • Введите 'в' или 'выход' для возврата в игру\n")  
 actions\_text.append("Подсказка: Номера предметов соответствуют их позициям в инвентаре", style="italic")  
  
 actions\_panel = Panel(  
 actions\_text,  
 title="Действия",  
 border\_style="red",  
 width=120  
 )  
  
 # Вывод всех панелей  
 console.print(Columns([status\_panel, equipment\_panel, inventory\_panel], expand=False))  
 console.print(actions\_panel)  
  
  
def display\_battle\_interface(player, enemy, ui=None, action\_manager=None):  
 *"""Отображает интерфейс боя с гарантированным применением цветов"""* clear\_screen()  
  
 # Левая панель - статус игрока  
 player\_stats = Text()  
 player\_stats.append(f"Имя: {player.name}\n", style="bold")  
 player\_stats.append(f"Уровень: {player.level}\n", style="cyan")  
  
 # Полоска здоровья как единый текст  
 health\_text = Text.from\_markup(get\_health\_bar(player.health\_points, player.max\_health\_points()))  
 player\_stats.append("Здоровье: ")  
 player\_stats.append(health\_text)  
 player\_stats.append("\n")  
  
 player\_stats.append(f"Атака: {player.attack\_power}\n")  
 player\_stats.append(f"Защита: {player.defence}\n")  
 player\_stats.append(f"Деньги: {player.money} монет\n", style="gold1")  
 player\_panel = Panel(player\_stats, title="Ваш герой", border\_style="yellow", width=40)  
  
 # Правая панель - статус врага  
 enemy\_stats = Text()  
 enemy\_stats.append(f"Имя: {enemy.name}\n", style="bold red")  
 enemy\_stats.append(f"Уровень: {enemy.level}\n", style="cyan")  
  
 # Полоска здоровья врага  
 enemy\_health\_text = Text.from\_markup(get\_health\_bar(enemy.health\_points, enemy.max\_health\_points()))  
 enemy\_stats.append("Здоровье: ")  
 enemy\_stats.append(enemy\_health\_text)  
 enemy\_stats.append("\n")  
  
 enemy\_stats.append(f"Атака: {enemy.attack\_power}\n")  
 enemy\_stats.append(f"Защита: {enemy.defence}\n")  
 enemy\_panel = Panel(enemy\_stats, title="Противник", border\_style="red", width=40)  
  
 # Выводим панели  
 console.print(Columns([player\_panel, enemy\_panel], expand=False, equal=False))  
  
 # Панель действий  
 actions = Text()  
 actions.append("Доступные действия:\n", style="bold underline")  
  
 if action\_manager:  
 # Группируем действия по типам для лучшего отображения  
 attack\_actions = []  
 other\_actions = []  
  
 for key in sorted(action\_manager.bindings.keys()):  
 action = action\_manager.bindings[key]  
 if action:  
 action\_name = {  
 "attack": "Обычная атака",  
 "escape": "Попытаться убежать",  
 "heal": "Использовать зелье лечения",  
 "strong\_attack": "Сильная атака",  
 "skill\_1": "Умение 1",  
 "skill\_2": "Умение 2"  
 }.get(action, action)  
  
 if "атака" in action\_name.lower() or "attack" in action.lower():  
 attack\_actions.append(f"{key} - {action\_name}")  
 else:  
 other\_actions.append(f"{key} - {action\_name}")  
  
 # Выводим атаки первой группой  
 if attack\_actions:  
 actions.append("Атаки:\n", style="bold yellow")  
 for action in attack\_actions:  
 actions.append(f"{action}\n")  
  
 # Затем остальные действия  
 if other\_actions:  
 actions.append("\nДругие действия:\n", style="bold green")  
 for action in other\_actions:  
 actions.append(f"{action}\n")  
 else:  
 # Стандартные действия (для обратной совместимости)  
 actions.append("1 - Обычная атака\n")  
 actions.append("2 - Попытаться убежать\n")  
 actions.append("3 - Использовать зелье лечения\n")  
 actions.append("4 - Сильная атака\n")  
  
 console.print(Panel(actions, title="Действия", border\_style="blue"))  
  
  
def show\_main\_menu():  
 *"""Отображает главное меню в стиле игрового интерфейса"""* clear\_screen()  
  
 # Создаем панель с командами  
 commands\_text = Text()  
 commands\_text.append("Доступные команды:\n", style="bold underline")  
 commands\_text.append("[С]тарт - начать новую игру\n")  
 commands\_text.append("[З]агрузить - загрузить игру\n")  
 commands\_text.append("[У]далить - удалить сохранение\n")  
 commands\_text.append("[УВ] - удалить все сохранения\n")  
 commands\_text.append("[В]ыход - выйти из игры\n")  
 commands\_panel = Panel(commands\_text, title="Главное меню", border\_style="green", width=38)  
  
 # Создаем панель статуса (пустую, так как нет героя)  
 status\_text = Text()  
 status\_text.append("Adventures of Heroes\n", style="bold")  
 status\_text.append("Версия: 0.5.2.2\n")  
 #status\_text.append("Автор: Ваше имя\n")  
 status\_panel = Panel(status\_text, title="Статус", border\_style="blue", width=38)  
  
 # Выводим панели рядом  
 console.print(Columns([commands\_panel, status\_panel], expand=True))  
  
  
def character\_menu(hero):  
 *"""Цикл меню характеристик героя"""* while True:  
 show\_character\_interface(hero)  
 command = input("\nВведите команду: ").strip().lower()  
  
 if command.startswith(('предмет ', 'п ')):  
 # Обработка команды использования предмета  
 try:  
 item\_num = int(command.split()[1]) - 1 # Получаем номер предмета из команды  
 hero.use\_item(number\_item=item\_num)  
 except (ValueError, IndexError):  
 console.print("[red]Ошибка: Неверный номер предмета[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
 elif command.startswith(('снять ', 'с ')):  
 # Обработка команды снятия предмета  
 try:  
 slot\_num = int(command.split()[1])  
 if 1 <= slot\_num <= 6:  
 slot\_names = list(hero.equipment.keys())  
 hero.remove\_item(slot=slot\_names[slot\_num - 1])  
 else:  
 console.print("[red]Ошибка: Номер слота должен быть от 1 до 6[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
 except (ValueError, IndexError):  
 console.print("[red]Ошибка: Неверный номер слота[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
 elif command in ["выход", "в"]:  
 break  
  
 else:  
 console.print("[red]Неверная команда![/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
  
  
def get\_main\_menu\_command():  
 *"""Получает команду в главном меню"""* while True:  
 show\_main\_menu()  
 command = input("\nВведите команду: ").strip().lower()  
  
 if command in ["старт", "с"]:  
 return "start"  
 elif command in ["загрузить", "з"]:  
 return "load"  
 elif command in ["удалить", "у"]:  
 return "delete"  
 elif command in ["удалить все", "ув"]:  
 return "delete\_all"  
 elif command in ["выход", "в"]:  
 return "exit"  
 else:  
 console.print("[red]Неверная команда. Попробуйте снова.[/red]")  
 time.sleep(1) # Задержка для чтения сообщения об ошибке  
  
def get\_commands\_panel():  
 *"""Создает панель с доступными командами"""* commands = Text()  
 commands.append("Доступные команды:\n", style="bold underline")  
 commands.append("[Г]ерой - характеристики и инвентарь\n")  
 commands.append("[С]охр - сохранить игру\n")  
 commands.append("[П]еремещ - сменить локацию\n")  
 commands.append("[Б]ой - начать бой (в боевой зоне)\n")  
 commands.append("[Т]орговец - торговля (в мирной зоне)\n")  
 commands.append("[КВ]есты - просмотр квестов\n")  
 commands.append("[НП]/[Н] - настройка панели действий\n") # Новая команда  
 commands.append("[В]ыход - выйти из игры")  
 return Panel(commands, title="Команды", border\_style="green")  
  
def clear\_screen():  
 *"""Очищает экран консоли"""* os.system('cls' if os.name == 'nt' else 'clear')  
  
def get\_player\_command(hero):  
 *"""Отображает интерфейс и получает команду от игрока с детализированной обработкой ошибок"""* clear\_screen()  
 status\_bar = get\_status\_bar(hero)  
 commands\_panel = get\_commands\_panel()  
  
 try:  
 # Пытаемся использовать стандартный вывод с Columns  
 console.print(Columns([status\_bar, commands\_panel], width=80))  
 except ZeroDivisionError:  
 # Специальная обработка для ZeroDivisionError  
 console.print("\n[red]ОШИБКА РЕНДЕРИНГА:[/red] [yellow]Проблема с разметкой колонок (деление на ноль)[/yellow]")  
 console.print("[yellow]Используется упрощенный интерфейс...[/yellow]\n")  
 console.print(status\_bar)  
 console.print(commands\_panel)  
 except Exception as e:  
 # Общая обработка всех других исключений  
 console.print(f"\n[red]КРИТИЧЕСКАЯ ОШИБКА:[/red] [yellow]{str(e)}[/yellow]")  
 console.print("[yellow]Тип ошибки:[/yellow]", type(e).\_\_name\_\_)  
 console.print("[yellow]Используется аварийный режим интерфейса...[/yellow]\n")  
 console.print(status\_bar)  
 console.print(commands\_panel)  
 finally:  
 # Этот блок выполнится в любом случае  
 console.print("\n[dim]Для продолжения введите команду:[/dim]", end=" ")  
  
 return input().strip().lower()  
  
def display\_interface(hero):  
 *"""Отображает игровой интерфейс с панелями бок о бок"""* os.system('cls' if os.name == 'nt' else 'clear')  
  
 # Левая панель - статус героя  
 status = Text()  
 status.append(f"Имя: {hero.name}\n", style="bold")  
 status.append(f"Уровень: {hero.level}\n")  
 status.append(f"Здоровье: {hero.health\_points}/{hero.max\_health\_points()}\n")  
 status.append(f"Опыт: {hero.experience}/{hero.exp\_base \* 2}\n")  
 status.append(f"Локация: {hero.location.name}\n")  
 status.append(f"Деньги: {hero.money} монет")  
 status\_panel = Panel(status, title="Статус героя", border\_style="blue", width=38)  
  
 # Правая панель - команды  
 commands = Text()  
 commands.append("Доступные команды:\n", style="bold underline")  
 commands.append("[Г]ерой - характеристики и инвентарь\n")  
 commands.append("[С]охр - сохранить игру\n")  
 commands.append("[П]еремещ - сменить локацию\n")  
  
 if hero.location.zone\_type == "combat":  
 commands.append("[Б]ой - начать бой\n")  
 else:  
 commands.append("[Т]орговец - торговля\n")  
  
 commands.append("[КВ]есты - просмотр квестов\n")  
 commands.append("[В]ыход - выйти из игры")  
 commands\_panel = Panel(commands, title="Команды", border\_style="green", width=38)  
  
 # Выводим панели рядом  
 console.print(Columns([commands\_panel, status\_panel ], expand=True))  
  
def get\_health\_color(hero):  
 *"""Возвращает цвет для здоровья в зависимости от процента"""* percent = hero.health\_points / hero.max\_health\_points()  
 if percent < 0.2: return "red"  
 if percent < 0.5: return "yellow"  
 return "green"  
  
def get\_status\_bar(hero):  
 *"""Улучшенная версия с цветовыми индикаторами"""* status\_text = Text()  
 status\_text.append(f"Имя: {hero.name}\n", style="bold")  
 status\_text.append(f"Уровень: {hero.level}\n", style="cyan")  
 status\_text.append("Здоровье: ", style="bold")  
 status\_text.append(f"{hero.health\_points}/{hero.max\_health\_points()}\n",  
 style=get\_health\_color(hero))  
 status\_text.append("Опыт: ", style="bold")  
 status\_text.append(f"{hero.experience}/{hero.exp\_base \* 2}\n", style="yellow")  
 status\_text.append("Локация: ", style="bold")  
 status\_text.append(f"{hero.location.name}\n", style="magenta")  
 status\_text.append("Деньги: ", style="bold")  
 status\_text.append(f"{hero.money} монет", style="gold1")  
 return Panel(status\_text, title="Статус героя", border\_style="blue")  
  
#Основные параметры предметов  
  
# Класс Квесты  
class Quest:  
 def \_\_init\_\_(self, quest\_id: int, name: str, description: str, target\_item\_id: int,  
 target\_amount: int, reward\_exp: int, reward\_money: int,  
 quest\_type: str, giver: str, is\_completed: bool = False,  
 current\_amount: int = 0):  
 self.id = quest\_id  
 self.name = name  
 self.description = description  
 self.target\_item\_id = target\_item\_id  
 self.target\_amount = target\_amount  
 self.reward\_exp = reward\_exp  
 self.reward\_money = reward\_money  
 self.quest\_type = quest\_type # single, daily, repeatable  
 self.giver = giver  
 self.is\_completed = is\_completed  
 self.current\_amount = current\_amount  
 self.completion\_date = None # Для ежедневных квестов  
  
 def \_\_str\_\_(self):  
 status = "[green]Завершён[/green]" if self.is\_completed else f"[yellow]{self.current\_amount}/{self.target\_amount}[/yellow]"  
 return f"{self.name} {status} - {self.description}"  
  
 def can\_be\_repeated(self) -> bool:  
 *"""Можно ли повторно получить этот квест"""* if self.quest\_type == "single":  
 return False  
 elif self.quest\_type == "daily":  
 # Проверяем, был ли квест завершен сегодня  
 if self.completion\_date and self.completion\_date.date() == datetime.now().date():  
 return False  
 return True  
 elif self.quest\_type == "repeatable":  
 return True  
 return False  
  
# Класс торговец  
class Merchant:  
 def \_\_init\_\_(self, name: str, items: list) -> None:  
 self.name = name  
 self.items = items # Список предметов, которые продает торговец  
  
 def show\_items(self):  
 console.print(f"{self.name} предлагает следующие товары:")  
 for index, item in enumerate(self.items):  
 console.print(f"{index + 1}. {item.name} - Цена: {item.stock\_price} монет")  
  
 def buy\_item(self, character: Character, item\_index: int):  
 if 0 <= item\_index < len(self.items):  
 item = deepcopy(self.items[item\_index]) # Создаем копию предмета  
 if character.money >= item.stock\_price:  
 character.money -= item.stock\_price  
 if character.add\_item(item):  
 console.print(f"{character.name} купил {item.name} у {self.name}.")  
 else:  
 console.print("[red]Не удалось добавить предмет в инвентарь![/red]")  
 character.money += item.stock\_price # Возвращаем деньги  
 else:  
 console.print("[red]Недостаточно денег![/red]")  
 else:  
 console.print("[red]Неверный индекс товара.[/red]")  
  
 def sell\_item(self, character: Character, item\_index: int):  
 if 0 <= item\_index < len(character.inventory):  
 item = character.inventory[item\_index]  
  
 # Для стакающихся предметов спрашиваем количество  
 if isinstance(item, StackableItem):  
 max\_sell = item.quantity  
 console.print(f"У вас есть {max\_sell} шт. {item.name}")  
 try:  
 sell\_count = int(input(f"Сколько хотите продать? (1-{max\_sell}): "))

if sell\_count < 1 or sell\_count > max\_sell:  
 console.print("[red]Неверное количество![/red]")  
 return  
 except ValueError:  
 console.print("[red]Введите число![/red]")  
 return  
  
 # Продаем указанное количество  
 sell\_price = (item.stock\_price // 2) \* sell\_count  
 character.money += sell\_price  
  
 if sell\_count == max\_sell:  
 character.inventory.pop(item\_index) # Удаляем весь стак  
 else:  
 item.quantity -= sell\_count # Уменьшаем количество  
  
 console.print(f"{character.name} продал {sell\_count} шт. {item.name} за {sell\_price} монет.")  
 else:  
 # Для нестакающихся предметов  
 sell\_price = item.stock\_price // 2  
 character.money += sell\_price  
 character.inventory.pop(item\_index)  
 console.print(f"{character.name} продал {item.name} за {sell\_price} монет.")  
 else:  
 console.print("[red]Неверный индекс товара.[/red]")  
  
  
  
  
  
  
#Принты для избавления от повторов  
def massage\_invalid\_command() -> str:  
 massage = (f"[dark\_olive\_green1]---------------------------------------------\n" + f"Неверная команда. Попробуйте ввести другую...\n" + f"---------------------------------------------[/dark\_olive\_green1]")  
 return console.print(massage)  
  
#Базы данных  
#База данных врагов  
list\_name\_orcs = [  
 'Внизуда','Азог', 'Балкмег', 'Болдог', 'Больг', 'Верховный Гоблин', 'Гольфимбул', 'Горбаг', 'Готмог', 'Гришнак',  
 'Лагдуф', 'Луг', 'Лугдуш', 'Лурц', 'Маухур', 'Музгаш', 'Нарзуг', 'Оркобал', 'Отрод', 'Радбуг',  
 'Снага', 'Углук', 'Уфтак', 'Фимбул', 'Шаграт', 'Шарку', 'Язнег'  
]  
#База данных предметов  
item\_database = [  
 StackableItem(name="Малое зелье лечения", effect="heal", effect\_heal=50, chance=33.3, stock\_price=10, id\_item=1,mob\_chances={"Азог":50, "Внизуда":33}), # 33.3  
 StackableItem(name="Среднее зелье лечения", effect="heal", effect\_heal=100, chance=10.0, stock\_price=20, id\_item=2, mob\_chances={"Азог":20,"Балкмег":30}),# 10  
 StackableItem(name="Большое зелье лечения", effect="heal", effect\_heal=200, chance=5.0, stock\_price=50, id\_item=3, mob\_chances={"Азог":10,"Балкмег":18}),  
 StackableItem(name="Жемчужина", effect="quest", effect\_heal=0, chance=50.0, stock\_price=25, id\_item=12, mob\_chances={"Внизуда": 50}),  
 Equipment(name="Шлем рыцаря", slot="Голова", effect="defence", effect\_value=5, chance=5.0, stock\_price=100, id\_item=4),  
 Equipment(name="Кираса рыцаря", slot="Тело", effect="defence", effect\_value=10, chance=5.0, stock\_price=200, id\_item=5,mob\_chances={"Балкмег":7}),  
 Equipment(name="Перчатки силы", slot="Руки", effect="attack", effect\_value=3, chance=5.0, stock\_price=75, id\_item=6),  
 Equipment(name="Сапоги ловкости", slot="Ноги", effect="defence", effect\_value=3, chance=5.0, stock\_price=75, id\_item=7),  
 Equipment(name="Меч воина", slot="Оружие", effect="attack", effect\_value=10, chance=5.0, stock\_price=150, id\_item=8),  
 Equipment(name="Плащ теней", slot="Плащ", effect="defence", effect\_value=7, chance=5.0, stock\_price=100, id\_item=9,mob\_chances={"Балкмег":7}),# 5  
 Equipment(name="Старые перчатки", slot="Руки", effect="defence", effect\_value=1, chance=20.0, stock\_price=8, id\_item=10, mob\_chances={"Внизуда":10}),  
 Equipment(name="Старые сапоги", slot="Ноги", effect="defence", effect\_value=2, chance=2.0, stock\_price=8, id\_item=11, mob\_chances={"Внизуда":0}),  
 # Item(name="Большое зелье лечения", effect="heal", effect\_heal=200, chance=5.0, stock\_price=50, id\_item=3) # 5  
]  
  
  
  
#Отдельные функции  
  
def spawn\_mob(location: Location):  
 if location.name == "Храм":  
 level = 1 # Константа! Монстры только первого уровня  
 name = list\_name\_orcs[0] # Имя орка из храма (Внизуда)  
 allowed\_item\_ids = [1, 2, 6, 7, 10, 11, 12] # Предметы, которые могут выпадать от монстров в храме [1, 2, 6, 7, 10, 11]  
 money = random.randint(5, 10) # Генерация денег для монстра в храме  
 elif location.name == "Зачарованный лес":  
 level = random.randint(5, 7) # уровни монстров для "Зачарованный лес"  
 name = list\_name\_orcs[2] # Имя орка из зачарованного леса (Балкмег)  
 allowed\_item\_ids = [1, 2, 3, 9, 4] # Предметы, которые могут выпадать от монстров в лесу [1, 2, 3, 9, 4]  
 money = random.randint(10, 20) # Генерация денег для монстра в лесу  
 elif location.name == "Безлюдная пустыня":  
 level = random.randint(10, 13) # уровни монстров для "Безлюдная пустыня"  
 name = list\_name\_orcs[1] # Имя орка из пустыни (Азог)  
 allowed\_item\_ids = [2, 3, 5, 8, 9] # Предметы, которые могут выпадать от монстров в пустыне [2, 3, 5, 8, 9]  
 money = random.randint(50, 90) # Генерация денег для монстра в пустыне  
 else:  
 return None # Если локация не распознана  
  
 # Создаем нового монстра с учетом уровня и денег  
 new\_spawn\_mob = Mob(name=name, level=level, item\_database=item\_database, allowed\_item\_ids=allowed\_item\_ids)  
 new\_spawn\_mob.money = money # Устанавливаем сгенерированное количество денег  
  
 return new\_spawn\_mob  
  
# Функции алаки и спелов  
def try\_escape(character: Character, enemy: Character) -> bool:  
 *"""Попытка убежать из боя с 30% шансом, если здоровье меньше половины"""* if character.health\_points < character.max\_health\_points() / 2:  
 return random.random() < 0.3  
 return False  
  
  
def use\_healing(character: Character) -> bool:  
 *"""Попытка использовать зелье лечения"""* for i, item in enumerate(character.inventory):  
 if isinstance(item, StackableItem) and item.effect == "heal":  
 character.use\_item(number\_item=i)  
 return True  
 return False  
  
  
def strong\_attack(character: Character, enemy: Character) -> int:  
 *"""Сильная атака с увеличенным уроном"""* damage = character.attack\_power \* 1.5 # Увеличенный урон  
 damage = damage \* (100 - enemy.defence) / 100  
 return round(damage)  
  
  
def get\_input\_with\_timeout(prompt, timeout):  
 *"""Получаем ввод с таймаутом, работает в Windows и Unix"""* print(prompt, end='', flush=True)  
  
 if 'msvcrt' in sys.modules:  
 # Реализация для Windows  
 start\_time = time.time()  
 input\_text = []  
 while (time.time() - start\_time) < timeout:  
 if msvcrt.kbhit():  
 char = msvcrt.getwch()  
 if char == '\r': # Enter  
 print()  
 return ''.join(input\_text)  
 elif char == '\x08': # Backspace  
 if input\_text:  
 input\_text.pop()  
 print('\b \b', end='', flush=True)  
 else:  
 input\_text.append(char)  
 print(char, end='', flush=True)  
 time.sleep(0.05)  
 print()  
 return None  
 else:  
 # Реализация для Unix  
 import select  
 ready, \_, \_ = select.select([sys.stdin], [], [], timeout)  
 if ready:  
 return sys.stdin.readline().strip()  
 return None  
  
  
def fight\_turn(player, enemy, ui=None, action\_manager=None):  
 *"""Один ход боя с новым интерфейсом"""* display\_battle\_interface(player, enemy, ui, action\_manager)  
  
 # Получаем действие игрока с таймером  
 action = get\_input\_with\_timeout("Выберите действие: ", TURN\_TIME)  
  
 # Если ввод пустой или None - считаем это обычной атакой  
 if action is None or action.strip() == "":  
 action = "1" # Дефолтное действие  
 console.print("[yellow]Автоматическая атака.[/yellow]")  
 time.sleep(0.3)  
 else:  
 action = action.strip() # Удаляем лишние пробелы  
  
 enemy\_killed = False  
  
 # Определяем тип действия  
 if action\_manager:  
 action\_type = action\_manager.get\_action(action)  
 else:  
 # Стандартные действия для обратной совместимости  
 action\_type = {  
 "1": "attack",  
 "2": "escape",  
 "3": "heal",  
 "4": "strong\_attack"  
 }.get(action)  
  
 # Обработка действий  
 if action\_type == "attack":  
 player.attack(target=enemy)  
 if not enemy.is\_alive():  
 enemy\_killed = True  
 elif action\_type == "escape":  
 if try\_escape(player, enemy):  
 console.print("[green]Вам удалось сбежать![/green]")  
 time.sleep(2)  
 return True  
 else:  
 console.print("[red]Не удалось сбежать![/red]")  
 time.sleep(1)  
 elif action\_type == "heal":  
 if use\_healing(player):  
 console.print("[green]Вы использовали зелье лечения![/green]")  
 time.sleep(1)  
 else:  
 console.print("[red]У вас нет зелий лечения![/red]")  
 time.sleep(1)  
 elif action\_type == "strong\_attack":  
 damage = strong\_attack(player, enemy)  
 enemy.got\_damage(damage=damage)  
 console.print(f"[yellow]Вы наносите сильный удар на {damage} урона![/yellow]")  
 time.sleep(1)  
 if not enemy.is\_alive():  
 enemy\_killed = True  
 else:  
 console.print("[red]Неизвестное действие![/red]")  
 time.sleep(1)  
  
 # Если враг жив и игрок не лечился - враг атакует  
 if enemy.is\_alive() and action\_type != "heal" and not enemy\_killed:  
 enemy.attack(target=player)  
 time.sleep(1)  
  
 return False  
  
  
def fight(\*, player, enemy, ui=None, action\_manager=None):  
 *"""Модифицированная функция боя с поддержкой настраиваемой панели действий"""* while True: # Основной цикл боя  
 display\_battle\_interface(player, enemy, ui, action\_manager)  
 console.print(f"[bold red]Начинается бой с {enemy.name}![/bold red]")  
 time.sleep(1.5)  
  
 while player.is\_alive() and enemy.is\_alive():  
 escaped = fight\_turn(player, enemy, ui, action\_manager)  
 if escaped:  
 return True  
  
 # Обработка результатов боя  
 if not enemy.is\_alive():  
 display\_battle\_interface(player, enemy, ui, action\_manager)  
 console.print(f"[bold green]Вы победили {enemy.name}![/bold green]")  
 console.print(f"[yellow3]Получено: {enemy.money} монет![/yellow3]")  
  
 # Показываем добычу  
 if enemy.inventory:  
 loot\_text = Text()  
 loot\_text.append("Полученная добыча:\n", style="bold")  
 for loot in enemy.inventory:  
 if isinstance(loot, StackableItem):  
 loot\_text.append(f"- {loot.name} (x{loot.quantity})\n", style="green")  
 else:  
 loot\_text.append(f"- {loot.name}\n", style="green")  
 console.print(Panel(loot\_text, title="Добыча", border\_style="green"))  
  
 # Панель действий после боя  
 actions\_text = Text()  
 actions\_text.append("Действия после боя:\n", style="bold underline")  
 actions\_text.append("[Enter] или [Б] - Начать новый бой\n")  
 actions\_text.append("[В] - Выйти из режима боя\n")  
 console.print(Panel(actions\_text, title="Выберите действие", border\_style="blue"))  
  
 # Ожидаем ввода пользователя  
 while True:  
 if 'msvcrt' in sys.modules: # Windows  
 if msvcrt.kbhit():  
 key = msvcrt.getch()  
 try:  
 key = key.decode('cp866').lower()  
 except UnicodeDecodeError:  
 continue  
  
 if key in ('\r', 'б'):  
 return False # Начать новый бой  
 elif key == 'в':  
 return True # Выйти из боя  
 else: # Unix  
 import select  
 if select.select([sys.stdin], [], [], 0)[0]:  
 key = sys.stdin.readline().strip().lower()  
 if key in ('', 'б'):  
 return False  
 elif key == 'в':  
 return True  
 time.sleep(0.1)  
  
 elif not player.is\_alive():  
 return True  
  
  
  
def fight\_with\_mob(ui=None, action\_manager=None):  
 *"""Функция боя с мобом с поддержкой настраиваемой панели"""* while True:  
 new\_mob = spawn\_mob(hero\_user.location)  
 if new\_mob:  
 if ui:  
 ui.add\_message(f"\n[bright\_red]Начинается бой с '{new\_mob.name}', {new\_mob.level} уровня[/bright\_red]")  
 ui.add\_message(f"[bright\_red]Здоровье врага: {new\_mob.health\_points}[/bright\_red]")  
 ui.update\_ui()  
 else:  
 console.print(f"\n[bright\_red]Начинается бой с '{new\_mob.name}', {new\_mob.level} уровня[/bright\_red]")  
 console.print(f"[bright\_red]Здоровье врага: {new\_mob.health\_points}[/bright\_red]")  
  
 should\_exit = fight(player=hero\_user, enemy=new\_mob, ui=ui, action\_manager=action\_manager)  
 if should\_exit:  
 break  
  
 # Обработка результатов боя  
 if not new\_mob.is\_alive():  
 hero\_user.money += new\_mob.money  
 if ui:  
 ui.add\_message(f"[yellow3]{hero\_user.name} получает: {new\_mob.money} монет![/yellow3]")  
 else:  
 console.print(f"[yellow3]{hero\_user.name} получает: {new\_mob.money} монет![/yellow3]")  
  
 for loot in new\_mob.inventory:  
 if isinstance(loot, StackableItem):  
 hero\_user.add\_item(item=loot)  
 if ui:  
 ui.add\_message(f"[yellow3]{hero\_user.name} получает: '{loot.name}'[/yellow3]")  
 else:  
 console.print(f"[yellow3]{hero\_user.name} получает: '{loot.name}'[/yellow3]")  
 else:  
 hero\_user.inventory.append(loot)  
 if ui:  
 ui.add\_message(f"[yellow3]{hero\_user.name} получает: '{loot}'[/yellow3]")  
 else:  
 console.print(f"[yellow3]{hero\_user.name} получает: '{loot}'[/yellow3]")  
 else:  
 if ui:  
 ui.add\_message("[red]Не удалось создать монстра.[/red]")  
 else:  
 console.print("[red]Не удалось создать монстра.[/red]")  
 break  
  
  
  
  
  
# Метод торговли  
def trade\_with\_merchant(hero, merchant):  
 *"""Основной цикл торговли с новым интерфейсом"""* pearl\_quest\_data = next((q for q in quest\_database if q["id"] == 1), None)  
 last\_message = ""  
  
 while True:  
 show\_trade\_interface(hero, merchant)  
  
 if last\_message:  
 console.print(f"\n[bold]{last\_message}[/bold]")  
 last\_message = ""  
 input("Нажмите Enter чтобы продолжить...")  
 continue  
  
 action = input("\nВведите команду: ").strip().lower()  
  
 if action.startswith(('купить ', 'к ')):  
 try:  
 item\_index = int(action.split()[1]) - 1  
 if 0 <= item\_index < len(merchant.items):  
 item = merchant.items[item\_index]  
 if hero.money >= item.stock\_price:  
 hero.money -= item.stock\_price  
 new\_item = deepcopy(item)  
 if hero.add\_item(new\_item):  
 last\_message = f"Вы купили: {item.name} за {item.stock\_price} монет"  
 else:  
 hero.money += item.stock\_price  
 last\_message = "Не удалось добавить предмет в инвентарь!"  
 else:  
 last\_message = "Недостаточно денег!"  
 else:  
 last\_message = "Неверный номер товара!"  
 except (ValueError, IndexError):  
 last\_message = "Используйте: 'купить [номер]'"  
  
 elif action.startswith(('продать ', 'п ')):  
 try:  
 item\_index = int(action.split()[1]) - 1  
 if 0 <= item\_index < len(hero.inventory):  
 item = hero.inventory[item\_index]  
 sell\_price = item.stock\_price // 2  
  
 if isinstance(item, StackableItem):  
 max\_sell = item.quantity  
 console.print(f"У вас есть {max\_sell} шт. {item.name}")  
 try:  
 sell\_count = int(input(f"Сколько хотите продать? (1-{max\_sell}): "))  
 if 1 <= sell\_count <= max\_sell:  
 hero.money += sell\_price \* sell\_count  
 if sell\_count == max\_sell:  
 hero.inventory.pop(item\_index)  
 else:  
 item.quantity -= sell\_count  
 last\_message = f"Продано {sell\_count} шт. {item.name} за {sell\_price \* sell\_count} монет"  
 else:  
 last\_message = "Неверное количество!"  
 except ValueError:  
 last\_message = "Введите число!"  
 else:  
 hero.money += sell\_price  
 hero.inventory.pop(item\_index)  
 last\_message = f"Вы продали: {item.name} за {sell\_price} монет"  
 else:  
 last\_message = "Неверный номер предмета!"  
 except (ValueError, IndexError):  
 last\_message = "Используйте: 'продать [номер]'"  
  
 elif action.startswith(('сдать ', 'с ')):  
 try:  
 merchant\_quests = [q for q in hero.active\_quests if q.giver == "Торговец"]  
 quest\_num = int(action.split()[1]) - 1  
 if 0 <= quest\_num < len(merchant\_quests):  
 quest = merchant\_quests[quest\_num]  
 if hero.is\_quest\_ready\_to\_complete(quest.id):  
 hero.complete\_quest(quest)  
 last\_message = f"Квест '{quest.name}' завершен!"  
 else:  
 last\_message = f"Не выполнены условия квеста! ({quest.current\_amount}/{quest.target\_amount})"  
 else:  
 last\_message = "Неверный номер квеста!"  
 except (ValueError, IndexError):  
 last\_message = "Используйте: 'сдать [номер]'"  
  
 elif action in ["квест", "кв"]:  
 if pearl\_quest\_data:  
 if hero.add\_quest(pearl\_quest\_data):  
 last\_message = f"Получен квест: '{pearl\_quest\_data['name']}'"  
 else:  
 last\_message = "Не удалось получить квест"  
 else:  
 last\_message = "Торговец сейчас не предлагает квестов"  
  
 elif action in ["выход", "в"]:  
 break  
  
 else:  
 last\_message = "Неверная команда!"  
  
#Функция перемещения персонажа  
def move\_character(ui=None):  
 *"""Функция перемещения персонажа с поддержкой нового интерфейса"""* if ui:  
 ui.add\_message("Выберите локацию для перемещения:")  
 for index, loc in enumerate(location\_database):  
 if loc.name != "Город" and hero\_user.location.name != "Город":  
 continue  
 ui.add\_message(f"{index + 1}. {loc.name} - {loc.description}")  
 ui.update\_ui()  
  
 choice = ui.get\_input("Введите номер локации для перемещения: ").strip()  
 else:  
 console.print("Выберите локацию для перемещения:")  
 for index, loc in enumerate(location\_database):  
 if loc.name != "Город" and hero\_user.location.name != "Город":  
 continue  
 console.print(f"{index + 1}. {loc.name} - {loc.description}")  
  
 choice = input("Введите номер локации для перемещения: ").strip()  
  
 if choice.isdigit():  
 choice\_index = int(choice) - 1  
 if 0 <= choice\_index < len(location\_database):  
 selected\_location = location\_database[choice\_index]  
 hero\_user.move\_to\_location(selected\_location)  
 if ui:  
 ui.add\_message(f"[green]Вы переместились в '{hero\_user.location.name}'![/green]")  
 else:  
 console.print(f"[green]Вы переместились в '{hero\_user.location.name}'![/green]")  
 else:  
 if ui:  
 ui.add\_message("[red]Ошибка: Неверный номер локации.[/red]")  
 else:  
 console.print("[red]Ошибка: Неверный номер локации.[/red]")  
 else:  
 if ui:  
 ui.add\_message("[red]Ошибка: Пожалуйста, введите корректный номер.[/red]")  
 else:  
 console.print("[red]Ошибка: Пожалуйста, введите корректный номер.[/red]")  
  
  
#Функции сохранения и загрузки  
def check\_file\_save(dict\_character: dict) -> bool:  
 if not isinstance(dict\_character, dict):  
 return False  
  
 # Проверка версии  
 version = dict\_character.get('version', 0)  
 if version != 4:  
 return False  
  
 # Проверка обязательных полей  
 required\_fields = {  
 'name': str,  
 'level': int,  
 'health\_points': int,  
 'attack\_power': int,  
 'defence': int,  
 'experience': int,  
 'exp\_base': int,  
 'count\_kill': int,  
 'location': str,  
 'money': int,  
 'now\_time': int  
 }  
  
 for field, field\_type in required\_fields.items():  
 if field not in dict\_character or not isinstance(dict\_character[field], field\_type):  
 return False  
  
 # Проверка инвентаря  
 if not isinstance(dict\_character.get('inventory'), list):  
 return False  
  
 for item in dict\_character['inventory']:  
 if not isinstance(item, dict) or 'id' not in item or 'quantity' not in item:  
 return False  
  
 # Проверка экипировки  
 if not isinstance(dict\_character.get('equipment'), list) or len(dict\_character['equipment']) != 6:  
 return False  
  
 return True  
  
def display\_saves(saves):  
 print("Список сохранений:")  
 for index, (save, status) in enumerate(saves, start=1):  
 if isinstance(save, dict): # Проверяем, что это словарь  
 name = save.get('name', 'Неизвестно')  
 level = save.get('level', 'Неизвестно')  
 time\_saved = save.get('now\_time', 0)  
 time\_formatted = datetime.fromtimestamp(time\_saved).strftime('%d-%m-%Y %H:%M:%S')  
 location = save.get('location', 'Неизвестно')  
 print(f"Ячейка сохранения № {index}: Имя: {name}, Уровень: {level}, Локация: {location}, Дата: {time\_formatted}, Статус: {status}")  
 else:  
 print(f"Ячейка сохранения № {index}: Статус: {status} (не удалось загрузить данные)")  
  
  
def convert\_old\_save(old\_save: dict) -> dict:  
 *"""Конвертирует старые сохранения (версии 3 и ниже) в новый формат (версия 4)"""* if not isinstance(old\_save, dict):  
 return old\_save  
  
 new\_save = old\_save.copy()  
 new\_save['version'] = 4 # Устанавливаем новую версию  
  
 # Конвертируем инвентарь  
 if 'inventory' in new\_save and isinstance(new\_save['inventory'], list):  
 new\_inventory = []  
 for item in new\_save['inventory']:  
 if isinstance(item, int): # Старый формат - только ID  
 new\_inventory.append({'id': item, 'quantity': 1})  
 elif isinstance(item, dict): # Уже новый формат  
 new\_inventory.append(item)  
 else: # Неизвестный формат  
 continue  
 new\_save['inventory'] = new\_inventory  
  
 # Конвертируем экипировку (если нужно)  
 if 'equipment' not in new\_save:  
 new\_save['equipment'] = [None] \* 6 # 6 слотов экипировки  
  
 return new\_save  
  
  
def save\_in\_file():  
 *"""Сохраняет игру с красивым уведомлением"""* try:  
 if not os.path.isdir("save"):  
 os.mkdir("save")  
  
 with open(file="save\\save.json", mode="a", encoding="utf-8") as file:  
 new\_dict\_for\_save = hero\_user.get\_all\_params\_for\_save()  
 json.dump(new\_dict\_for\_save, file, indent=4)  
 file.write("₽") # Разделитель сохранений  
  
 # Создаем красивую панель уведомления  
 success\_panel = Panel(  
 Text("Сохранение успешно завершено!\n", justify="center" ) + #style="white"  
 Text(f"Герой: {hero\_user.name}\n") +  
 Text(f"Уровень: {hero\_user.level}\n") +  
 Text(f"Локация: {hero\_user.location.name}\n") +  
 Text(f"Время сохранения: {datetime.now().strftime('%d.%m.%Y %H:%M')}"),  
 title="✓ Сохранение игры",  
 border\_style="gold1",  
 width=60  
 )  
  
 clear\_screen()  
 console.print(success\_panel)  
 console.print("\n[dim]Нажмите Enter чтобы продолжить...[/dim]")  
 input() # Ждём нажатия Enter  
  
 except Exception as e:  
 error\_panel = Panel(  
 Text(f"Ошибка сохранения!\n{str(e)}", style="bold red"),  
 title="Ошибка",  
 border\_style="red",  
 width=60  
 )  
 console.print(error\_panel)  
 console.print("\n[dim]Нажмите Enter чтобы продолжить...[/dim]")  
 input() # Ждём нажатия Enter  
  
  
def get\_list\_all\_saves():  
 list\_saves = []  
 try:  
 if not os.path.exists('save\\save.json'):  
 return list\_saves  
  
 with open('save\\save.json', 'r', encoding='utf-8') as file:  
 content = file.read().strip()  
 if not content:  
 return list\_saves  
  
 saves = content.split("₽")  
 for save\_str in saves:  
 if not save\_str.strip():  
 continue  
  
 try:  
 save\_data = json.loads(save\_str)  
 # Конвертируем старые сохранения в новый формат  
 if save\_data.get('version', 0) < 4:  
 save\_data = convert\_old\_save(save\_data)  
  
 # Проверяем сохранение  
 if check\_file\_save(save\_data):  
 list\_saves.append((save\_data, "OK"))  
 else:  
 list\_saves.append((save\_data, "Ошибка проверки"))  
 except json.JSONDecodeError:  
 list\_saves.append((save\_str, "Ошибка формата JSON"))  
 except Exception as e:  
 list\_saves.append((save\_str, f"Ошибка обработки: {str(e)}"))  
  
 except Exception as e:  
 console.print(f"[red]Ошибка при чтении файла сохранения: {str(e)}[/red]")  
  
 return list\_saves  
  
  
def download(database: list):  
 list\_saves = get\_list\_all\_saves()  
 if not list\_saves:  
 console.print("[yellow]Нет доступных сохранений.[/yellow]")  
 return None  
  
 # Показываем все сохранения с статусом  
 console.print("Доступные сохранения:")  
 valid\_count = 0  
 for index, (save, status) in enumerate(list\_saves, 1):  
 if isinstance(save, dict):  
 name = save.get('name', 'Неизвестно')  
 level = save.get('level', 0)  
 time\_str = datetime.fromtimestamp(save.get('now\_time', 0)).strftime('%d.%m.%Y %H:%M')  
 console.print(f"{index}. {name} (ур. {level}), {time\_str} - {status}")  
 if status == "OK":  
 valid\_count += 1  
  
 if valid\_count == 0:  
 console.print("[red]Нет корректных сохранений для загрузки.[/red]")  
 return None  
  
 while True:  
 try:  
 choice = input("Введите номер сохранения для загрузки (0 - отмена): ").strip()  
 if choice == '0':  
 return None  
  
 choice\_idx = int(choice) - 1  
 if 0 <= choice\_idx < len(list\_saves):  
 save\_data, status = list\_saves[choice\_idx]  
 if status == "OK":  
 hero = load\_hero\_user(dict\_param=save\_data, database=database)  
 if hero:  
 console.print("[green]Сохранение успешно загружено![/green]")  
 return hero  
 else:  
 console.print("[red]Не удалось загрузить героя.[/red]")  
 else:  
 console.print(f"[red]Нельзя загрузить это сохранение: {status}[/red]")  
 else:  
 console.print("[red]Неверный номер сохранения.[/red]")  
 except ValueError:  
 console.print("[red]Пожалуйста, введите число.[/red]")  
  
def load\_hero\_user(\*, dict\_param: dict, database: list):  
 # Проверяем версию сохранения  
 if dict\_param.get('version') != 4:  
 console.print("[red]Неверная версия сохранения[/red]")  
 return None  
  
 try:  
 # Создаем нового персонажа  
 hero = Human(name=dict\_param['name'], level=dict\_param['level'])  
 hero.health\_points = dict\_param['health\_points']  
 hero.attack\_power = dict\_param['attack\_power']  
 hero.defence = dict\_param['defence']  
 hero.experience = dict\_param['experience']  
 hero.exp\_base = dict\_param['exp\_base']  
 hero.count\_kill = dict\_param['count\_kill']  
 hero.money = dict\_param['money']  
 hero.class\_character = dict\_param['class\_character']  
  
 # Загружаем локацию  
 location\_name = dict\_param.get('location', 'Город')  
 hero.location = next(  
 (loc for loc in location\_database if loc.name == location\_name),  
 next(loc for loc in location\_database if loc.name == "Город") # fallback  
 )  
  
 # Загружаем инвентарь (новый формат)  
 hero.inventory = []  
 for item\_data in dict\_param.get('inventory', []):  
 item = next((i for i in database if i.id\_item == item\_data['id']), None)  
 if item:  
 if isinstance(item, StackableItem):  
 new\_item = deepcopy(item)  
 new\_item.quantity = item\_data.get('quantity', 1)  
 hero.inventory.append(new\_item)  
 else:  
 hero.inventory.append(deepcopy(item))  
  
 # Загружаем экипировку  
 equipment\_ids = dict\_param.get('equipment', [])  
 slot\_names = ["Голова", "Тело", "Руки", "Ноги", "Оружие", "Плащ"]  
 for i, item\_id in enumerate(equipment\_ids):  
 if item\_id is not None:  
 item = next((item for item in database if item.id\_item == item\_id), None)  
 if item:  
 hero.equipment[slot\_names[i]] = item  
 # Загружаем активные квесты  
 hero.active\_quests = []  
 for quest\_data in dict\_param.get('active\_quests', []):  
 # Находим квест в базе данных  
 quest\_template = next((q for q in quest\_database if q["id"] == quest\_data["id"]), None)  
 if quest\_template:  
 quest = Quest(  
 quest\_id=quest\_template["id"],  
 name=quest\_template["name"],  
 description=quest\_template["description"],  
 target\_item\_id=quest\_template["target\_item\_id"],  
 target\_amount=quest\_template["target\_amount"],  
 reward\_exp=quest\_template["reward\_exp"],  
 reward\_money=quest\_template["reward\_money"],  
 quest\_type=quest\_template["quest\_type"],  
 giver=quest\_template["giver"],  
 is\_completed=quest\_data.get("is\_completed", False),  
 current\_amount=quest\_data.get("current\_amount", 0)  
 )  
 # Восстанавливаем дату завершения  
 if quest\_data.get("completion\_date"):  
 quest.completion\_date = datetime.fromtimestamp(quest\_data["completion\_date"])  
 hero.active\_quests.append(quest)  
  
 # Загружаем завершенные квесты (только ID)  
 hero.completed\_quests = []  
 for quest\_id in dict\_param.get('completed\_quests', []):  
 quest\_template = next((q for q in quest\_database if q["id"] == quest\_id), None)  
 if quest\_template:  
 quest = Quest(  
 quest\_id=quest\_template["id"],  
 name=quest\_template["name"],  
 description=quest\_template["description"],  
 target\_item\_id=quest\_template["target\_item\_id"],  
 target\_amount=quest\_template["target\_amount"],  
 reward\_exp=quest\_template["reward\_exp"],  
 reward\_money=quest\_template["reward\_money"],  
 quest\_type=quest\_template["quest\_type"],  
 giver=quest\_template["giver"],  
 is\_completed=True  
 )  
 hero.completed\_quests.append(quest)  
 hero.update\_stats()  
 return hero  
  
 except Exception as e:  
 console.print(f"[red]Ошибка при загрузке персонажа: {str(e)}[/red]")  
 return None  
  
def delete\_specific\_save(save\_index: int):  
 saves = get\_list\_all\_saves()  
 if 0 <= save\_index < len(saves):  
 confirmation = input("Вы уверены, что хотите удалить это сохранение? (да/нет): ")  
 if confirmation.lower() == 'да':  
 with open('save\\save.json', 'r+', encoding='utf-8') as file:  
 content = file.read()  
 saves = content.split("₽") # Разбиваем по разделителю  
 del saves[save\_index] # Удаляем выбранное сохранение  
 file.seek(0)  
 file.truncate() # Очищаем файл  
 file.write("₽".join(saves)) # Записываем оставшиеся сохранения  
 console.print("[yellow3]Сохранение успешно удалено.[/yellow3]")  
 else:  
 console.print("[yellow3]Удаление сохранения отменено.[/yellow3]")  
 else:  
 console.print("[red]Ошибка: Неверный номер ячейки.[/red]")  
  
def prompt\_for\_save\_deletion():  
 display\_saves(get\_list\_all\_saves()) # Показываем сохранения  
 save\_index\_input = input("Введите номер ячейки для удаления: ").strip() # Убираем пробелы  
  
 if not save\_index\_input.isdigit(): # Проверяем, является ли ввод числом  
 console.print("[red]Ошибка: Пожалуйста, введите корректный номер ячейки.[/red]")  
 return # Возвращаемся в основное меню  
  
 save\_index = int(save\_index\_input) - 1 # Преобразуем ввод в индекс  
 delete\_specific\_save(save\_index) # Удаляем сохранение  
  
def delete\_all\_saves():  
 confirmation = input("Вы уверены, что хотите удалить все сохранения? Это действие нельзя отменить! (да/нет): ")  
 if confirmation.lower() == 'да':  
 with open('save\\save.json', 'w', encoding='utf-8') as file:  
 file.write("") # Очищаем файл  
 console.print("[yellow3]Все сохранения успешно удалены.[/yellow3]")  
 else:  
 console.print("[yellow3]Удаление всех сохранений отменено.[/yellow3]")  
  
#Здесь функция игры  
def game() -> None:  
 global action\_manager # Используем глобальный менеджер действий  
  
 try:  
 while hero\_user.is\_alive():  
 command = get\_player\_command(hero\_user)  
  
 # Бой  
 if command in ["бой", "б"] and hero\_user.location.zone\_type == "combat":  
 fight\_with\_mob(action\_manager=action\_manager)  
  
 # Квесты  
 elif command in ["квесты", "кв"]:  
 hero\_user.show\_quests()  
  
 # Торговец  
 elif command in ["торговец", "т"] and hero\_user.location.zone\_type == "peaceful":  
 merchant\_items = [item\_database[0], item\_database[1], item\_database[4]]  
 merchant = Merchant(name="Торговец", items=merchant\_items)  
 trade\_with\_merchant(hero\_user, merchant)  
  
 # Перемещение  
 elif command in ["перемещение", "п"]:  
 move\_character()  
  
 # Сохранение игры  
 elif command in ["сохранить", "с"]:  
 save\_in\_file()  
  
 # Настройка панели действий  
 elif command in ["настройка панели", "нп", "н"]:  
 configure\_action\_panel(action\_manager)  
  
 # Выход из игры  
 elif command in ["выход", "в"]:  
 confirmation = input(  
 "Вы уверены, что хотите выйти? Весь несохраненный прогресс будет утерян! (да/нет):"  
 ).strip().lower()  
 if confirmation in ["да", "д"]:  
 print("Выход из игры.")  
 break  
 else:  
 console.print("[yellow3]Вы остаетесь в игре.[/yellow3]")  
 continue  
  
 # Характеристики героя  
 elif command in ["герой", "г"]:  
 character\_menu(hero\_user)  
  
 # Неизвестная команда  
 else:  
 console.print("[red]Неверная команда![/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
 except Exception as e:  
 console.print(f"[red]Произошла ошибка в игре: {str(e)}[/red]")  
 import traceback  
 traceback.print\_exc()  
 input("\nНажмите Enter чтобы продолжить...")  
  
 finally:  
 # Сохраняем настройки панели действий при выходе  
 try:  
 action\_manager.save\_config()  
 console.print("[green]Настройки панели действий сохранены.[/green]")  
 except Exception as e:  
 console.print(f"[red]Ошибка при сохранении настроек панели: {str(e)}[/red]")  
  
 # Финальное сообщение  
 console.print(  
 f"\n[bright\_cyan]────────────────────────────────────────────[/bright\_cyan]"  
 f"\n[bright\_cyan] Имя героя: '{hero\_user.name}', Уровень: {hero\_user.level} "  
 f"\n[bright\_cyan] Ждем Вашего возвращения!"  
 f"\n[bright\_cyan]────────────────────────────────────────────[/bright\_cyan]"  
 )  
 input("\nНажмите Enter чтобы продолжить...")  
  
#Здесь конец функции игры  
  
# Основной блок игры  
console.print(f"Добро пожаловать в игру\n\n[red]--- Adventures of Heroes ---\n[/red]")  
  
while True:  
 menu\_command = get\_main\_menu\_command()  
  
 # В основном игровом цикле:  
 if menu\_command == "start":  
 hero\_user = character\_creation\_flow()  
 game() # Запускаем игру с созданным персонажем  
  
 elif menu\_command == "load":  
 # Код загрузки игры  
 hero\_user = download(database=item\_database)  
 if hero\_user:  
 hero\_user.class\_character = hero\_user.get\_class\_hero()  
 console.print(f"[yellow3]--------------------------------------------------\n"  
 f"Успешно загружено\n"  
 f"--------------------------------------------------[/yellow3]\n")  
 game()  
 #except Exception as e: # Ловим все исключения  
 # console.print(f"[red]----------- КАКАЯ-ТО ОШИБКА ------------------\n""--------------------------------------------------\n""Откройте папку с игрой.\n""Рядом с файлом 'adventures\_of\_heroes.\_.\_.exe'\n""должна быть папка 'save'\n""В папке 'save' должен быть файл 'save.json' \n""Или файл с сохранением был испорчен\n"f"Ошибка: {str(e)}\n""--------------------------------------------------[/red]") # Выводим текст ошибки для отладки  
  
 elif menu\_command == "delete":  
 prompt\_for\_save\_deletion()  
 elif menu\_command == "delete\_all":  
 delete\_all\_saves()  
 elif menu\_command == "exit":  
 console.print(f"[bright\_cyan]-----------------\n---Конец игры----\n-----------------[/bright\_cyan]")  
 break

# game/character.py  
from rich.columns import Columns  
from rich.panel import Panel  
from rich.text import Text  
from rich.console import Console  
from rich.box import ROUNDED  
import time  
from datetime import datetime  
from copy import deepcopy  
from datetime import datetime  
import random  
from rich.panel import Panel  
from rich.text import Text  
#импорты из соседних файлов  
from .items import Item, Equipment, StackableItem  
from .utils import generate\_inventory, clear\_screen  
from .quests import Quest  
from .quests import quest\_database  
from .locations import Location  
  
  
#Разметка цветом  
console = Console()  
  
  
#Основные параметры классов  
class Character:  
 def \_\_init\_\_(self, name: str, level: int) -> None:  
 self.name = name  
 self.level = level  
 self.health\_points = self.base\_health\_points \* level  
 self.attack\_power = self.base\_attack\_power \* level  
 self.defence = self.base\_defence \* level  
 self.experience = 0  
 self.exp\_base = 100  
 self.count\_kill = 0  
 self.location = None # Устанавливаем начальную локацию  
 self.class\_character = None  
 self.inventory = []  
 self.equipment = {  
 "Голова": None,  
 "Тело": None,  
 "Руки": None,  
 "Ноги": None,  
 "Оружие": None,  
 "Плащ": None,  
 }  
 self.money = 0 # Новое поле для хранения денег  
  
 def \_\_str\_\_(self):  
 return f"Class: {self.get\_class\_hero()}. Name:'{self.name}', level: {self.level} HP: {self.health\_points}, Money: {self.money}"  
  
 def hero\_inventory(self) -> None:  
 print(f"---------------------\n"  
 f"Содержимое инвентаря:\n"  
 f"---------------------"  
 )  
 i = 1  
 for inventory\_item in self.inventory:  
 print(f"Ячейка № {i}: '{inventory\_item}'")  
 i += 1  
  
 def show\_character\_and\_inventory(self) -> None:  
 # Вывод характеристик героя  
 stats = (  
 f"Имя Вашего героя: '{self.name}', [yellow3]Уровень:[/yellow3][yellow3] {self.level}[/yellow3]\n"  
 f"Здоровье: {self.health\_points}/{self.max\_health\_points()}\n"  
 f"Защита героя: {self.defence}\n"  
 f"Атака героя: {self.attack\_power}\n"  
 f"Уровень: {self.level}\n"  
 f"Опыт героя: {self.experience} из {self.exp\_base \* 2} до следующего уровня\n"  
 f"Количество убитых врагов: {self.count\_kill}\n"  
 f"Локация: {self.location.name}\n"  
 f"Деньги: {self.money} монет" # Отображение денег  
 )  
  
 # Вывод снаряжения в формате ячеек  
 equipment = (  
 f"Снаряжение Вашего героя:\n"  
 f"1. Голова: {self.equipment.get('Голова', 'None')}\n"  
 f"2. Тело: {self.equipment.get('Тело', 'None')}\n"  
 f"3. Руки: {self.equipment.get('Руки', 'None')}\n"  
 f"4. Ноги: {self.equipment.get('Ноги', 'None')}\n"  
 f"5. Оружие: {self.equipment.get('Оружие', 'None')}\n"  
 f"6. Плащ: {self.equipment.get('Плащ', 'None')}\n"  
 )  
  
 # Вывод инвентаря с учетом количества стакающихся предметов  
 inventory\_items = []  
 for i, item in enumerate(self.inventory, 1):  
 if isinstance(item, StackableItem):  
 inventory\_items.append(f"{i}. {item.name} (x{item.quantity})")  
 else:  
 inventory\_items.append(f"{i}. {item.name}")  
  
 inventory\_str = "\n".join(inventory\_items) if inventory\_items else "Инвентарь пуст."  
  
 console.print(f"[green]{stats}[/green]\n[blue]{equipment}[/blue]\n[blue]Инвентарь:\n{inventory\_str}[/blue]")  
  
 def get\_class\_hero(self) -> str:  
 return self.\_\_class\_\_.\_\_name\_\_  
 def get\_class\_hero\_rus(self) -> str:  
 class\_hero = ""  
 if self.get\_class\_hero() == "Human":  
 class\_hero = "Человек"  
 elif self.get\_class\_hero() == "Warrior":  
 class\_hero = "Воин"  
 elif self.get\_class\_hero() == "Mage":  
 class\_hero = "Маг"  
 return class\_hero  
  
 def is\_alive(self) -> bool:  
 return self.health\_points > 0  
  
 def got\_damage(self, \*, damage: int) -> None:  
 damage = damage \* (100 - self.defence) / 100  
 damage = round(damage)  
 self.health\_points -= damage  
  
 def gain\_experience(self, \*, target: "Character") -> None:  
 if not (target.is\_alive()):  
 self.experience += target.max\_health\_points() \* 4  
  
  
 def level\_up(self, exp\_base: int):  
 exp\_base = exp\_base \* 2  
 if self.experience >= exp\_base:  
 self.level += 1  
 self.health\_points = self.base\_health\_points \* self.level  
 self.attack\_power = self.base\_attack\_power \* self.level  
 self.defence = self.base\_defence \* self.level  
 self.exp\_base = exp\_base  
 self.experience = self.experience - exp\_base  
 console.print(f"[bright\_cyan]{self.name} получает опыт и повышает уровень до {self.level}[/bright\_cyan]")  
  
 def attack(self, \*, target: "Character") -> None:  
 print(f"{self.name} атакует {target.name}")  
 target.got\_damage(damage=self.attack\_power)  
 if target.is\_alive():  
 print(f"{self.name}, HP={self.health\_points} | {target.name}, HP={target.health\_points}")  
 else:  
 print(f"{target.name} погибает!")  
 # Убрал gain\_experience и level\_up из этого метода, так как они теперь обрабатываются в fight()  
 self.count\_kill += 1  
 if isinstance(target, Mob):  
 self.money += target.money  
  
  
 def max\_health\_points(self):  
 return self.base\_health\_points \* self.level  
  
 def add\_item(self, item: Item) -> bool:  
 # Проверка на нулевое количество  
 if isinstance(item, StackableItem) and item.quantity <= 0:  
 return False  
 # Для нестакающихся предметов  
 if not isinstance(item, StackableItem):  
 if len(self.inventory) < MAX\_INVENTORY\_SIZE: # Максимальный размер инвентаря константа в нчале  
 self.inventory.append(item)  
 return True  
 return False  
  
 # Для стакающихся предметов  
 for existing\_item in self.inventory:  
 if (isinstance(existing\_item, StackableItem) and  
 existing\_item.id\_item == item.id\_item and  
 existing\_item.quantity < existing\_item.max\_stack):  
  
 # Сколько можно добавить в этот стак  
 space\_left = existing\_item.max\_stack - existing\_item.quantity  
 add\_amount = min(item.quantity, space\_left)  
  
 existing\_item.quantity += add\_amount  
 item.quantity -= add\_amount  
  
 if item.quantity <= 0:  
 return True  
  
 # Если остались предметы или нет подходящего стака  
 while item.quantity > 0 and len(self.inventory) < MAX\_INVENTORY\_SIZE:  
 new\_stack = deepcopy(item)  
 new\_stack.quantity = min(item.quantity, item.max\_stack)  
 self.inventory.append(new\_stack)  
 item.quantity -= new\_stack.quantity  
  
 return item.quantity == 0  
  
 def use\_item(self, \*, number\_item: int) -> None:  
 *"""Использует предмет из инвентаря с удобным интерфейсом"""* try:  
 # Проверка корректности номера предмета  
 if number\_item < 0 or number\_item >= len(self.inventory):  
 raise IndexError("Номер предмета выходит за границы инвентаря")  
  
 item = self.inventory[number\_item]  
  
 # Обработка экипировки  
 if isinstance(item, Equipment):  
 slot = item.slot  
  
 # Если в слоте уже есть предмет - сообщаем о снятии  
 if self.equipment[slot] is not None:  
 old\_item = self.equipment[slot]  
 self.inventory.append(old\_item)  
 console.print(f"\n[yellow3]Снят предмет: {old\_item.name}[/yellow3]")  
  
 # Экипируем новый предмет  
 self.equipment[slot] = item  
 self.inventory.pop(number\_item)  
 self.update\_stats()  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Экипирован предмет: {item.name} ({slot})[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 # Обработка стакающихся предметов (зелий и т.д.)  
 elif isinstance(item, StackableItem):  
 # Применяем эффект  
 if item.effect == "heal":  
 heal\_amount = min(item.effect\_heal, self.max\_health\_points() - self.health\_points)  
 self.health\_points += heal\_amount  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Использовано: {item.name}[/green]"  
 f"\n[green]Восстановлено: {heal\_amount} здоровья[/green]"  
 f"\n[green]Осталось: {item.quantity - 1} шт.[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 # Уменьшаем количество  
 item.quantity -= 1  
  
 # Если предмет закончился - удаляем из инвентаря  
 if item.quantity <= 0:  
 self.inventory.pop(number\_item)  
  
 # Обработка обычных предметов  
 else:  
 if item.effect == "heal":  
 heal\_amount = min(item.effect\_heal, self.max\_health\_points() - self.health\_points)  
 self.health\_points += heal\_amount  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Использовано: {item.name}[/green]"  
 f"\n[green]Восстановлено: {heal\_amount} здоровья[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 # Удаляем предмет после использования  
 self.inventory.pop(number\_item)  
  
 # Ждем подтверждения от игрока  
 input("\nНажмите Enter чтобы продолжить...")  
  
 except IndexError as e:  
 console.print(f"[red]Ошибка: {str(e)}[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
 except Exception as e:  
 console.print(f"[red]Неизвестная ошибка при использовании предмета: {str(e)}[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
 def \_use\_equipment(self, item: Equipment, slot\_index: int) -> None:  
 *"""Вспомогательный метод для использования экипировки"""* slot = item.slot  
  
 # Если в слоте уже есть предмет - сообщаем о снятии  
 if self.equipment[slot] is not None:  
 old\_item = self.equipment[slot]  
 self.inventory.append(old\_item)  
 console.print(f"\n[yellow3]Снят предмет: {old\_item.name}[/yellow3]")  
  
 # Экипируем новый предмет  
 self.equipment[slot] = item  
 self.inventory.pop(slot\_index)  
 self.update\_stats()  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Экипирован предмет: {item.name} ({slot})[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 def \_use\_stackable\_item(self, item: StackableItem, item\_index: int) -> None:  
 *"""Вспомогательный метод для использования стакающихся предметов"""* # Применяем эффект  
 if item.effect == "heal":  
 heal\_amount = min(item.effect\_heal, self.max\_health\_points() - self.health\_points)  
 self.health\_points += heal\_amount  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Использовано: {item.name}[/green]"  
 f"\n[green]Восстановлено: {heal\_amount} здоровья[/green]"  
 f"\n[green]Осталось: {item.quantity - 1} шт.[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 # Уменьшаем количество  
 item.quantity -= 1  
  
 # Если предмет закончился - удаляем из инвентаря  
 if item.quantity <= 0:  
 self.inventory.pop(item\_index)  
  
 def \_use\_regular\_item(self, item: Item, item\_index: int) -> None:  
 *"""Вспомогательный метод для обычных предметов"""* if item.effect == "heal":  
 heal\_amount = min(item.effect\_heal, self.max\_health\_points() - self.health\_points)  
 self.health\_points += heal\_amount  
  
 console.print(  
 f"\n[green]====================================[/green]"  
 f"\n[green]Использовано: {item.name}[/green]"  
 f"\n[green]Восстановлено: {heal\_amount} здоровья[/green]"  
 f"\n[green]====================================[/green]"  
 )  
  
 # Удаляем предмет после использования  
 self.inventory.pop(item\_index)  
  
 def remove\_item(self, slot: str) -> None:  
 *"""Снимает предмет экипировки с выводом информации"""* if slot in self.equipment and self.equipment[slot] is not None:  
 removed\_item = self.equipment[slot]  
 self.equipment[slot] = None  
 self.inventory.append(removed\_item)  
 self.update\_stats()  
  
 console.print(  
 f"\n[yellow3]====================================[/yellow3]"  
 f"\n[yellow3]Снят предмет: {removed\_item.name}[/yellow3]"  
 f"\n[yellow3]Из слота: {slot}[/yellow3]"  
 f"\n[yellow3]====================================[/yellow3]"  
 )  
  
 # Ждем подтверждения от игрока  
 input("\nНажмите Enter чтобы продолжить...")  
 else:  
 console.print(f"[red]Ошибка: В слоте '{slot}' нет экипированного предмета.[/red]")  
 input("\nНажмите Enter чтобы продолжить...")  
  
 def discard\_item(self, number\_item: int) -> None:  
 if 0 <= number\_item < len(self.inventory):  
 discarded\_item = self.inventory.pop(number\_item)  
 console.print(f"[yellow3]Вы выбросили '{discarded\_item}' из инвентаря.[/yellow3]")  
 else:  
 console.print("[red]Ошибка: Неверный номер предмета.[/red]")  
  
 def get\_all\_params\_for\_save(self) -> dict:  
 save\_hero = {  
 'version': 4, # Обновляем версию на 4  
 'name': self.name,  
 'level': self.level,  
 'health\_points': self.health\_points,  
 'attack\_power': self.attack\_power,  
 'defence': self.defence,  
 'experience': self.experience,  
 'exp\_base': self.exp\_base,  
 'count\_kill': self.count\_kill,  
 'location': self.location.name if isinstance(self.location, Location) else "Город",  
 'class\_character': self.class\_character,  
 'inventory': self.\_get\_inventory\_for\_save(),  
 'equipment': self.\_get\_equipment\_for\_save(), # Используем новый метод  
 'money': self.money, # Сохраняем количество денег  
 'active\_quests': [{  
 'id': q.id,  
 'current\_amount': q.current\_amount,  
 'is\_completed': q.is\_completed,  
 'completion\_date': q.completion\_date.timestamp() if q.completion\_date else None  
 } for q in self.active\_quests],  
 'completed\_quests': [q.id for q in self.completed\_quests],  
 'now\_time': round(time.time())  
 }  
 return save\_hero  
  
 def get\_list\_id\_item\_from\_save(self, items) -> list:  
 i = 0  
 inventory\_from\_save = []  
 for \_ in self.inventory:  
 inventory\_from\_save.append(self.inventory[i].id\_item)  
 i += 1  
 return [item.id\_item if item else None for item in items]  
  
 def \_get\_inventory\_for\_save(self) -> list:  
 inventory\_data = []  
 for item in self.inventory:  
 if isinstance(item, StackableItem):  
 inventory\_data.append({  
 'id': item.id\_item,  
 'quantity': item.quantity  
 })  
 else:  
 inventory\_data.append({  
 'id': item.id\_item,  
 'quantity': 1  
 })  
 return inventory\_data  
  
 def \_get\_equipment\_for\_save(self) -> list:  
 *"""Возвращает список ID предметов экипировки для сохранения"""* equipment\_data = []  
 for slot in ["Голова", "Тело", "Руки", "Ноги", "Оружие", "Плащ"]:  
 item = self.equipment.get(slot)  
 if item:  
 equipment\_data.append(item.id\_item)  
 else:  
 equipment\_data.append(None)  
 return equipment\_data  
  
 def update\_stats(self):  
 self.attack\_power = self.base\_attack\_power \* self.level  
 self.defence = self.base\_defence \* self.level  
 for item in self.equipment.values():  
 if item:  
 if item.effect == "attack":  
 self.attack\_power += item.effect\_value  
 elif item.effect == "defence":  
 self.defence += item.effect\_value  
  
 def set\_location(self, location) -> None:  
 self.location = location  
  
 def move\_to\_location(self, new\_location, ui=None):  
 if self.location.name == "Город":  
 self.set\_location(new\_location)  
 if ui:  
 ui.add\_message(f"[green]Вы переместились в '{self.location.name}'![/green]")  
 else:  
 console.print(f"[green]Вы переместились в '{self.location.name}'![/green]")  
 elif new\_location.name == "Город":  
 self.set\_location(new\_location)  
 if ui:  
 ui.add\_message(f"[green]Вы вернулись в 'Город'![/green]")  
 else:  
 console.print(f"[green]Вы вернулись в 'Город'![/green]")  
 else:  
 if ui:  
 ui.add\_message(f"[red]Вы можете перемещаться только в 'Город' из '{self.location.name}'![/red]")  
 else:  
 console.print(f"[red]Вы можете перемещаться только в 'Город' из '{self.location.name}'![/red]")  
  
 def get\_location(self) -> str:  
 return str(self.location) if isinstance(self.location, Location) else "Неизвестно"  
  
 def add\_quest(self, quest\_data: dict) -> bool:  
 *"""Добавляет квест из базы данных"""* # Проверяем, можно ли получить квест  
 if self.level < quest\_data["required\_level"]:  
 console.print(  
 f"[red]Ваш уровень слишком низок для этого квеста (требуется: {quest\_data['required\_level']})[/red]")  
 return False  
  
 # Проверяем выполнены ли требуемые квесты  
 for req\_quest\_id in quest\_data["required\_quests"]:  
 if not any(q.id == req\_quest\_id and q.is\_completed for q in self.completed\_quests):  
 console.print(f"[red]Вы не выполнили необходимые предварительные квесты[/red]")  
 return False  
  
 # Проверяем ограничение по локации  
 if quest\_data["location\_restriction"] and self.location.id\_loc != quest\_data["location\_restriction"]:  
 console.print(f"[red]Этот квест можно получить только в определённой локации[/red]")  
 return False  
  
 # Проверяем, есть ли уже такой квест в активных или завершенных  
 existing\_active = next((q for q in self.active\_quests if q.id == quest\_data["id"]), None)  
 existing\_completed = next((q for q in self.completed\_quests if q.id == quest\_data["id"]), None)  
  
 # Для одноразовых квестов проверяем, был ли он уже выполнен  
 if quest\_data["quest\_type"] == "single" and existing\_completed:  
 console.print("[red]Этот квест уже был выполнен и больше недоступен[/red]")  
 return False  
  
 # Если квест уже активен, проверяем можно ли его повторить  
 if existing\_active:  
 if existing\_active.quest\_type == "single":  
 console.print("[red]Этот квест уже активен[/red]")  
 return False  
 elif not existing\_active.can\_be\_repeated():  
 console.print("[red]Вы уже выполнили этот квест сегодня[/red]")  
 return False  
  
 # Создаем экземпляр квеста  
 new\_quest = Quest(  
 quest\_id=quest\_data["id"],  
 name=quest\_data["name"],  
 description=quest\_data["description"],  
 target\_item\_id=quest\_data["target\_item\_id"],  
 target\_amount=quest\_data["target\_amount"],  
 reward\_exp=quest\_data["reward\_exp"],  
 reward\_money=quest\_data["reward\_money"],  
 quest\_type=quest\_data["quest\_type"],  
 giver=quest\_data["giver"]  
 )  
  
 self.active\_quests.append(new\_quest)  
 console.print(f"[yellow3]Получен новый квест: '{new\_quest.name}'[/yellow3]")  
 console.print(f"[yellow3]Описание: {new\_quest.description}[/yellow3]")  
 return True  
  
 def complete\_quest(self, quest: Quest) -> None:  
 *"""Завершает квест и выдает награду"""* # Удаляем требуемые предметы из инвентаря  
 if quest.target\_item\_id:  
 target\_amount = quest.target\_amount  
 # Проходим по инвентарю в обратном порядке для безопасного удаления  
 for i in range(len(self.inventory) - 1, -1, -1):  
 item = self.inventory[i]  
 if item.id\_item == quest.target\_item\_id:  
 if item.quantity <= target\_amount:  
 target\_amount -= item.quantity  
 self.inventory.pop(i)  
 else:  
 item.quantity -= target\_amount  
 target\_amount = 0  
 if target\_amount == 0:  
 break  
  
 # Выдаем награду  
 self.experience += quest.reward\_exp  
 self.money += quest.reward\_money  
 quest.is\_completed = True  
 quest.completion\_date = datetime.now()  
  
 # Перемещаем квест в завершенные (для одноразовых)  
 if quest.quest\_type == "single":  
 self.active\_quests.remove(quest)  
 self.completed\_quests.append(quest)  
 else:  
 quest.current\_amount = 0  
 quest.is\_completed = False  
  
 console.print(f"[bright\_green]====================================[/bright\_green]")  
 console.print(f"[bright\_green]Квест '{quest.name}' завершен![/bright\_green]")  
 console.print(f"[bright\_green]Награда: {quest.reward\_exp} опыта и {quest.reward\_money} монет[/bright\_green]")  
 console.print(f"[bright\_green]====================================[/bright\_green]")  
  
 if quest.quest\_type == "single":  
 self.active\_quests.remove(quest)  
 self.completed\_quests.append(quest)  
 else:  
 quest.current\_amount = 0  
 quest.is\_completed = False  
  
 def show\_quests(self) -> None:  
 *"""Показывает активные и завершенные квесты"""* try:  
 clear\_screen()  
 active\_quests\_panel = self.\_create\_quests\_panel(self.active\_quests, "Активные квесты", "yellow")  
 completed\_quests\_panel = self.\_create\_quests\_panel(self.completed\_quests, "Завершенные квесты", "green")  
  
 console.print(Columns([active\_quests\_panel, completed\_quests\_panel], expand=True))  
 input("\nНажмите Enter чтобы вернуться в меню...")  
 except Exception as e:  
 console.print(f"[red]Ошибка при показе квестов: {str(e)}[/red]")  
 input("Нажмите Enter чтобы продолжить...")  
  
 def \_create\_quests\_panel(self, quests: list, title: str, color: str) -> Panel:  
 *"""Создает панель с квестами"""* quests\_text = Text()  
  
 if not quests:  
 quests\_text.append("Нет квестов", style="italic")  
 else:  
 for i, quest in enumerate(quests, 1):  
 # Для активных квестов показываем прогресс  
 if title == "Активные квесты":  
 status = Text("Готово к сдаче!", style="green") if self.is\_quest\_ready\_to\_complete(  
 quest.id) else f"{quest.current\_amount}/{quest.target\_amount}"  
 quests\_text.append(f"{i}. {quest.name} - ")  
 quests\_text.append(status)  
 quests\_text.append("\n")  
 quests\_text.append(f" Описание: {quest.description}\n")  
 quests\_text.append(f" Награда: {quest.reward\_exp} опыта и {quest.reward\_money} монет\n"  
 f" Для сдачи квеста введите у {quest.giver} '[с]дать {i}'")  
 # Для завершенных - просто список  
 else:  
 quests\_text.append(f"{i}. {quest.name}\n", style=color)  
  
 return Panel(quests\_text, title=title, border\_style=color)  
  
 def is\_quest\_ready\_to\_complete(self, quest\_id: int) -> bool:  
 *"""Проверяет, можно ли завершить квест"""* quest = next((q for q in self.active\_quests if q.id == quest\_id), None)  
 if not quest:  
 return False  
  
 # Проверяем, есть ли нужные предметы в инвентаре  
 if quest.target\_item\_id:  
 total = sum(item.quantity for item in self.inventory  
 if item.id\_item == quest.target\_item\_id)  
 return total >= quest.target\_amount  
 return True  
  
 def check\_quest\_progress(self, item\_id: int, amount: int = 1) -> None:  
 *"""Проверяет прогресс по квестам при получении предмета"""* for quest in self.active\_quests:  
 if quest.target\_item\_id == item\_id and not quest.is\_completed:  
 quest.current\_amount += amount  
  
 # Проверяем, нужно ли автоматическое завершение  
 quest\_data = next((q for q in quest\_database if q["id"] == quest.id), None)  
 if quest\_data and quest\_data.get("auto\_complete", False):  
 if quest.current\_amount >= quest.target\_amount:  
 self.complete\_quest(quest)  
  
#Все подклассы  
class Human(Character):  
 base\_health\_points = 100  
 base\_attack\_power = 10  
 base\_defence = 5  
 base\_inventory = []  
class Warrior(Human):  
 base\_health\_points = 200  
 base\_attack\_power = 20  
 base\_defence = 10  
class Mage(Human):  
 base\_health\_points = 100  
 base\_attack\_power = 40  
 base\_defence = 6  
class Mob(Character):  
 base\_health\_points = 100 # test  
 base\_attack\_power = 8  
 base\_defence = 3  
  
 def \_\_init\_\_(self, \*, name: str, level: int, item\_database: list, allowed\_item\_ids: list) -> None:  
 super().\_\_init\_\_(name, level)  
 self.inventory = generate\_inventory(item\_database, allowed\_item\_ids, mob\_name=name) # Передаем имя монстра  
 self.money = random.randint(5, 20) # Генерация случайного количества денег для монстра  
  
#class Mob\_mini(Mob)

# game/items.py  
from copy import deepcopy  
#Основные параметры предметов  
class Item:  
 def \_\_init\_\_(self, name: str, effect: str, effect\_heal: int, chance: float, stock\_price: int, id\_item: int, mob\_chances: dict = None) -> None:  
 self.name = name  
 self.effect = effect  
 self.chance = chance  
 self.effect\_heal = effect\_heal  
 self.stock\_price = stock\_price  
 self.id\_item = id\_item  
 self.mob\_chances = mob\_chances if mob\_chances else {} # Словарь шансов для разных монстров  
  
 def \_\_str\_\_(self):  
 return self.name  
# Подкласс для стакающихся предметов  
class StackableItem(Item):  
 def \_\_init\_\_(self, name: str, effect: str, effect\_heal: int, chance: float, stock\_price: int,  
 id\_item: int, quantity: int = 1, max\_stack: int = 20, mob\_chances: dict = None) -> None:  
 super().\_\_init\_\_(name, effect, effect\_heal, chance, stock\_price, id\_item, mob\_chances)  
 self.quantity = quantity  
 self.max\_stack = max\_stack  
  
 def \_\_str\_\_(self):  
 return f"{self.name} (x{self.quantity})" # Этот метод уже правильный  
# Основные параметры экипировки  
class Equipment(Item):  
 def \_\_init\_\_(self, name: str, slot: str, effect: str, effect\_value: int, chance: float, stock\_price: int, id\_item: int, mob\_chances: dict = None) -> None:  
 super().\_\_init\_\_(name, effect, 0, chance, stock\_price, id\_item, mob\_chances)  
 self.slot = slot  
 self.effect\_value = effect\_value

# game/locations.py  
  
class Location:  
 def \_\_init\_\_(self, name: str, description: str, danger\_level: int, zone\_type: str, id\_loc: int) -> None:  
 self.name = name  
 self.description = description  
 self.danger\_level = danger\_level  
 self.zone\_type = zone\_type # Новый параметр для типа зоны  
 self.id\_loc = id\_loc  
  
 def \_\_str\_\_(self):  
 return f"{self.name}: {self.description} (Уровень опасности: {self.danger\_level}, Тип зоны: {self.zone\_type})"  
  
  
  
#База данных локаций  
location\_database = [  
 Location(name="Город", description="Место, полное жизни и возможностей.", danger\_level=1, zone\_type="peaceful", id\_loc="1"),  
 Location(name="Зачарованный лес", description="Лес, полный магии и тайн. Уровни монстров: (5-7)", danger\_level=3, zone\_type="combat", id\_loc="2"),  
 Location(name="Безлюдная пустыня", description="Широкие песчаные дюны и отсутствие жизни.Уровни монстров: (10-13)", danger\_level=4, zone\_type="combat", id\_loc="3"),  
 Location(name="Храм", description="Древний храм, хранящий множество секретов.Уровни монстров: (1)", danger\_level=2, zone\_type="combat", id\_loc="4"),  
  
]

# game/quests.py  
from datetime import datetime  
  
class Quest:  
 def \_\_init\_\_(self, quest\_id: int, name: str, description: str, target\_item\_id: int,  
 target\_amount: int, reward\_exp: int, reward\_money: int,  
 quest\_type: str, giver: str, is\_completed: bool = False,  
 current\_amount: int = 0):  
 self.id = quest\_id  
 self.name = name  
 self.description = description  
 self.target\_item\_id = target\_item\_id  
 self.target\_amount = target\_amount  
 self.reward\_exp = reward\_exp  
 self.reward\_money = reward\_money  
 self.quest\_type = quest\_type # single, daily, repeatable  
 self.giver = giver  
 self.is\_completed = is\_completed  
 self.current\_amount = current\_amount  
 self.completion\_date = None # Для ежедневных квестов  
  
 def \_\_str\_\_(self):  
 status = "[green]Завершён[/green]" if self.is\_completed else f"[yellow]{self.current\_amount}/{self.target\_amount}[/yellow]"  
 return f"{self.name} {status} - {self.description}"  
  
 def can\_be\_repeated(self) -> bool:  
 *"""Можно ли повторно получить этот квест"""* if self.quest\_type == "single":  
 return False  
 elif self.quest\_type == "daily":  
 # Проверяем, был ли квест завершен сегодня  
 if self.completion\_date and self.completion\_date.date() == datetime.now().date():  
 return False  
 return True  
 elif self.quest\_type == "repeatable":  
 return True  
 return False  
  
  
# База данных квестов  
quest\_database = [  
 {  
 "id": 1,  
 "name": "Жемчужный сбор",  
 "description": "Принесите 10 жемчужин, которые падают с монстра 'Внизуда'",  
 "target\_item\_id": 12, # ID жемчужины  
 "target\_amount": 10,  
 "reward\_exp": 200,  
 "reward\_money": 500,  
 "giver": "Торговец",  
 "quest\_type": "single", # Типы: single (одноразовый), daily (ежедневный), repeatable (повторяемый)  
 "required\_level": 1,  
 "required\_quests": [], # ID квестов, которые нужно выполнить перед этим  
 "location\_restriction": None, # Можно указать ID локации, где можно получить квест  
 "auto\_complete": False, # False - нужно сдавать вручную, True - завершается автоматически  
 },  
 # Здесь можно добавлять другие квесты по аналогии  
]

# game/utils.py  
import random  
import os  
from copy import deepcopy  
from rich.console import Console  
  
  
#импорты из соседних файлов  
from .items import Item, StackableItem  
  
console = Console()  
  
def generate\_inventory(item\_database: list, allowed\_item\_ids: list, max\_item=5, mob\_name: str = None):  
 inventory = []  
 filtered\_items = [item for item in item\_database if item.id\_item in allowed\_item\_ids]  
  
 for \_ in range(max\_item):  
 item\_template = random.choice(filtered\_items)  
 chance = item\_template.mob\_chances.get(mob\_name, item\_template.chance)  
  
 if random.uniform(0, 100) < chance:  
 if isinstance(item\_template, StackableItem):  
 # Создаем новый экземпляр с quantity=1  
 new\_item = deepcopy(item\_template)  
 new\_item.quantity = 1  
 inventory.append(new\_item)  
 else:  
 inventory.append(deepcopy(item\_template))  
  
 return inventory  
  
def clear\_screen():  
 *"""  
 Универсальная очистка экрана с многоуровневой обработкой ошибок.  
 Пробует методы в порядке приоритета:  
 1. Через rich.console (самый надежный)  
 2. Стандартный os.system  
 3. Резервный вывод пустых строк  
 """* try:  
 # Пытаемся очистить через rich (предпочтительный способ)  
 console.clear()  
 except Exception as rich\_error:  
 try:  
 # Если rich не сработал, пробуем стандартный способ  
 os.system('cls' if os.name == 'nt' else 'clear')  
 except Exception as os\_error:  
 try:  
 # Для Jupyter/особых случаев  
 from IPython.display import clear\_output  
 clear\_output(wait=True)  
 except:  
 # Последний резерв - выводим много пустых строк  
 print('\n' \* 100)  
 # Логируем ошибки, если нужно  
 if 'rich\_error' in locals():  
 console.print(f"[yellow]Rich clear error: {rich\_error}[/yellow]")  
 if 'os\_error' in locals():  
 console.print(f"[yellow]OS clear error: {os\_error}[/yellow]")