МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВВГУ»)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАЛИЗА ДАННЫХ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЁТ
ПО ФИНАЛЬНОЙ ЛАБОРАТОРНОЙ РАБОТЕ

По дисциплине
«Информатика и программирование»

Студент
гр. Бин-25-2 _____ И.А. Маклаков
Ассистент
Преподавателя _____ М.В. Водяницкий

Владивосток 2026

Задание

Техническое задание - Текстовая RPG-игра

Вы работаете программистом в небольшой японской компании на заре игровой индустрии. Компания разрабатывает свою первую экспериментальную игру - текстовую RPG, которая должна запускаться прямо в консоли и погружать игрока в атмосферу подземелий, опасностей и развития персонажа

Ваша задача - реализовать прототип игры, который демонстрирует основные игровые механики: характеристики персонажа, бои, прокачку, инвентарь и случайные события

1. Общая идея программы

Программа представляет собой консольную текстовую RPG, в которой игрок:

* создаёт персонажа (выбор расы)

* получает случайные характеристики в рамках выбранной расы

* исследует подземелье, состоящее из случайных комнат

* сражается с врагами, находит предметы и улучшает персонажа

* повышает уровень и распределяет очки характеристик

* принимает решения, влияющие на дальнейший путь

Игра работает в пошаговом режиме и управляется вводом команд с клавиатуры

2. Создание персонажа

2.1 Выбор расы

В начале игры пользователь выбирает расу персонажа (например):

* Человек

* Эльф

* Дворф

Каждая раса задаёт диапазоны генерации характеристик

2.2 Характеристики персонажа

Характеристики генерируются случайным образом при создании персонажа, но в допустимых пределах для выбранной расы

Пример набора характеристик (можно расширять):

HP - здоровье

Attack - сила атаки

Defense - защита

Agility - ловкость (влияет на уклонение)

Height - рост

Weight - вес

Допускается, что некоторые характеристики влияют друг на друга (например, рост и вес влияют на уклонение или скорость)

3. Опыт и уровни

* Персонаж получает опыт за победу над врагами
* При накоплении нужного количества опыта повышается уровень
* Каждый новый уровень даёт очки прокачки

3.1 Прокачка характеристик

Игрок может распределять очки вручную между характеристиками

Пример:

* +1 к атаке
* +2 к HP
* +1 к ловкости

Распределение очков выполняется в комнатах отдыха

4. Инвентарь и экипировка

4.1 Инвентарь

Инвентарь хранит предметы:

* зелья (лечение и др.)
* монеты
* оружие
* прочие предметы

Игрок может:

* просматривать инвентарь
* использовать предметы
* выбрасывать любые предметы

4.2 Экипировка

В инвентаре должны быть отдельные слоты:

* оружие
* броня

Экипированные предметы влияют на характеристики персонажа

5. Подземелье и комнаты

5.1 Структура подземелья

* Игра начинается в подземелье
* Подземелье состоит из комнат
* После каждой комнаты игрок выбирает путь:

  * налево
  * направо

Развилка есть после каждой комнаты

5.2 Типы комнат

Комнаты генерируются случайно:

* Боевая комната - бой с врагом
* Комната отдыха - без событий
* Комната с сундуком - предметы или золото

Возможны комбинации:

* слева враг, справа сундук
* оба врага
* обе комнаты отдых

5.3 Видимость комнат

Перед выбором направления игрок:

* иногда знает, что находится дальше
* иногда не знает (темно, неизвестно)

Информация о видимости определяется случайно.

6. Враги и сложность

* Враги генерируются случайно
* У врагов есть характеристики (HP, атака, защита и т.д.)
* С каждым этажом подземелья сложность возрастает
* Каждые N комнат или действий происходит переход на новый этаж

7. Боевая система

Бой происходит в пошаговом режиме:

Пример действий игрока:

* атаковать
* использовать предмет
* попытаться уклониться

Учитываются:

* характеристики игрока
* экипировка
* случайные факторы (уклонение, критический удар)

8. Предметы и добыча

* Враги и сундуки могут давать:
  * зелья

  * оружие
  * другие предметы
* Полученные предметы добавляются в инвентарь
* При нехватке места игрок решает, что выбросить

9. Хранение данных

Допускается (но не обязательно):

* сохранение состояния игры в файл
* использование формата JSON для хранения:
  * характеристик персонажа
  * инвентаря
  * текущего этажа

10. Пример работы программы (фрагмент)

Выберите расу:

1 - Человек

2 - Эльф

3 - Дворф

> 2

Ваш персонаж создан!

HP: 85

ATK: 12

DEF: 6

AFI: 14

Вы входите в подземелье...

Перед вами развилка.

(1) Слева: ???

(2) Справа: Комната отдыха

Куда пойти?

> 1

12. Ограничения и требования

* Программа консольная
* Управление через текстовое меню и ввод команд
* Язык программирования - не ограничен
* Код должен быть читаемым и логически структурированным, можно делить на разные файлы

Содержание

1 Выполнение работы

1.1 Создание json файлов с данными об игре

Всего было сделано 10 файлов расширения json, немного подробнее о каждом:

1) Character.json - в данном файле будут сохранены текущие характеристики персонажа, его снаряжение, инвентарь, золото и очки навыков

2) Main.json - в данном файле содержатся настройки игры, а именно максимальный размер инвентаря, количество комнат на этаже, множитель характеристик монстров за этаж, сколько опыта требует каждый уровень, количество очков навыков за уровень, начальный этаж

3) Races.json - в данном файле прописаны название рас и их диапазон характеристик: сила, ловкость, живучесть, рост и вес

4) Monsters.json - в данном файле прописаны характеристики монстров, их локации, предметы, которые с них падают, и шансы на их получение

5) Rooms.json - в данном файле содержатся данные о возможный перекрёстках, информации о том какая комната будет в направлении справа и слева, шансы на попадания в конкретный перекрёсток, вещи которые содержатся в комнатах сокровищницах

6) Weapons.json - в данном файле лежат данные обо всём доступном, на данный момент, оружии, а также его описание

7) Armors.json - в данном файле лежат данные обо всей доступной, на данный момент, броне, а также его описание

8) Potions.json - в данном файле лежит информация обо всех доступных на данный момент зельях и их описание

9) miscItems.json - в данном файле содержатся данные об неиспользуемых предметах

10) Saves.json - в данном файле будет сохранение персонажа, этажа, сложности и прочего

Каждый предмет, сущность, комната, развилка имеют свой уникальный id.

1.2 Создание файла в которой будет код игры

Создаётся файл DungeonGame с расширением py, все дальнейшие действия происходят в нём.

1.3 Импорт библиотек и использование данных из json

Импортируются библиотеки: json, random, os, math, «from typing import Dict, List, Any, Optional». Создаются переменные character_file, races_file, monsters_file... misc_items_file набранные печатными буквами. Они содержат в себе данные из json файлов по данному пути.

1.4 Класс Game

1.4.1 def __init__()

В данной функции происходит загрузка основных данных из json файлов. Функция изображена на Рисунке 1.

```
def __init__(self):
    self.player = self.load_json(CHARACTER_FILE)
    self.races = self.load_json(RACES_FILE)["races"]
    self.monsters = self.load_json(MONSTERS_FILE)["monsters"]
    self.armors = {a["id"]: a for a in self.load_json(ARMORS_FILE)}
    self.weapons = {w["id"]: w for w in self.load_json(WEAPONS_FILE)}
    self.potions = {p["id"]: p for p in self.load_json(POTIONS_FILE)}
    self.misc_items = {m["id"]: m for m in self.load_json(MISC_ITEMS_FILE)["misc_items"]}
    self.rooms = self.load_json(ROOMS_FILE)["rooms"]
    self.settings = self.load_json(MAIN_FILE)["game_settings"]

    self.current_room = None
    self.current_floor = self.settings["start_floor"]
    self.rooms_visited = 0
    self.game_over = False
```

Рисунок 1 - Листинг кода функции __init__

Пояснение работы функции:

1) self это ссылка внутри внутри класса на конкретный объект этого класса

2) self.player, self.races, self.monsters - чтение данных их переменных раздела 1.2

3) self.armors, self.weapons и так далее - чтение данных и преобразование в словари по ключу и значению(ключ - id, значение - весь остаток)

4) self.rooms - список комнат

5) self.setting - настройки игры

6) self.current_room - текущая комната

7) self.current_floor - текущий этаж

8) self.rooms_visited - счётчик посещённых комнат

9) self.game_over - флажок обозначающий завершена игра или нет

Так устроена функция __init__ в коде

1.4.2 def load_json()

Данная функция реализует безопасную загрузку данных из json файлов с кэшированием. Функция изображена на Рисунке 2.

```
def load_json(self, path: str, key: Optional[str] = None) -> Any:
    if not hasattr(self, "_cache"):
        self._cache = {}
    cache_key = f"{path}_{key}"
    if cache_key in self._cache:
        return self._cache[cache_key]

    try:
        with open(path, "r", encoding="utf-8") as f:
            data = json.load(f)
        if key is None:
            result = data
        else:
            if key not in data:
                raise KeyError(f"В JSON-файле {path} отсутствует ключ '{key}'")
            result = data[key]
        self._cache[cache_key] = result
        return result
    except (FileNotFoundError, json.JSONDecodeError, KeyError) as e:
        print(f"Ошибка при загрузке {path}: {e}")
        return {}
    except Exception as e:
        print(f"Непредвиденная ошибка: {e}")
        raise
```

Рисунок 2 - Листинг кода для функции load_json

Пояснение работы функции:

1) Проверяет наличие ключа _cache, в случае отсутствия создаёт его, затем формирует его уникальный идентификатор, а после проверяет есть ли уже данные и загружает их, вместо повторного чтения файла

2) Открывает файл в кодировке utf-8, преобразовывает json в объекты Python, в случае если не указан по какому ключу искать, возвращает весь json файл. Также в нём прописана обработка ошибок, а именно: нет указанного ключа, неправильный синтаксис json файла, не найден json файл. В конце обрабатывает остальные ошибки что не были учтены

Таким образом выполняется безопасная загрузка json файлов

1.4.3 def update_armor_from_defense()

Данная функция выполняет перерасчёт брони по защите персонажа. Функция изображена на Рисунке 3.

```
def update_armor_from_defense(self):
    defense_bonus = self.player["stats"]["defense"] // 2
    self.player["armor"] = self.player.get("base_armor", 0) + defense_bonus
```

Рисунок 3 - Листинг функции update_armor_from_defense

Здесь переменная defense_bonus получает целочисленное значение брони из метода self.player(защита) делённое на 2, а после записывается в параметр armor у персонажа.

1.4.4 get_dodge_chance()

Данная функция выполняет расчёт шанса уклонения. Функция изображена на Рисунке 4.

```
def get_dodge_chance(self):
    return 0.05 + (self.player["stats"]["agility"] // 3) * 0.005
```

Рисунок 4 - Листинг кода для функции get_dodge_chance

Здесь при вызове функции она возвращает шанс ловкости по формуле 0.05(5%) плюс целочисленное значение ловкости персонажа от деления на 3, умноженное на 0.005(0.5%)

1.4.5 def get_crit_chance()

Данная функция выполняет расчёт шанса критического удара. Функция изображена на Рисунке 5.

```
def get_crit_chance(self):
    return 0.05 + (self.player["stats"]["agility"] // 2) * 0.005
```

Рисунок 5 - Листинг кода для функции get_crit_chance

Здесь при вызове функции она возвращает шанс критического удара по формуле 0.05(5%) плюс целочисленное значение ловкости персонажа от деления на 2, умноженное на 0.005(0.5%)

1.4.6 def create_character()

Данная функция выполняет процесс создания персонажа. Функция изображена на Рисунке 6.

```
def create_character(self):
        print("=== Создание персонажа ===")
        self.player["name"] = input("Имя персонажа: ").strip()

        print("\nВыберите расу:")
        print("1 – Человек")
        print("2 – Эльф")
        print("3 – Дворф")

        race_choice = input("\n> ").strip()

        race_map = {
            "1": "human",
            "2": "elf",
            "3": "dwarf"
        }

        if race_choice not in race_map:
            print("Обнаружен неккоректный ввод. По умолчанию выбран Человек.")
            race_id = "human"
        else:
            race_id = race_map[race_choice]

        selected_race = next((r for r in self.races if r["id"] == race_id), None)
        if not selected_race:
            print("Ошибка: данные о выбранной расе отсутствуют!")
            return

        self.player["race"] = selected_race["name"]
        print(f"\nВы выбрали: {self.player["race"]}")

        if "stats" not in self.player:
            self.player["stats"] = {}

        for stat in ["strength", "agility", "vitality", "Weight", "Height"]:
            min_val, max_val = selected_race["attributes"][stat]
            self.player["stats"][stat] = random.randint(min_val, max_val)

        weight = self.player["stats"]["Weight"]
        height = self.player["stats"]["Height"]
        agility_bonus = max(0, (height - 120) / 10 - weight / 20)
        self.player["stats"]["agility"] += int(agility_bonus)

        self.player["max_health"] = self.player["stats"]["vitality"] * 5
        self.player["health"] = self.player["max_health"]
        self.player["damage"] = self.player["stats"]["strength"] + 1

        self.player["base_armor"] = 0
        self.player["stats"]["defense"] = ( self.player["stats"]["strength"] + self.player["stats"]["vitality"] ) // 3

        self.update_armor_from_defense()

        self.player["stat_points"] = 0


        print(f"\nПерсонаж создан! Здоровье: {self.player["health"]}, Броня: {self.player["armor"]}")
```

Рисунок 6 - Листинг кода для функции create_character

Игроку выводится процесс создания персонажа: сначала программа просит ввести его имя, после происходит выбор расы(при неверном вводе будет выбрана раса человек), по выбору пользователя персонаж получает значения характеристик из диапазонов в настройке рас. После этого идёт расчёт значений здоровья, урона, защиты и брони. Максимальное здоровья рассчитывается как значение живучести умноженное на 5; урон равен значению силы плюс 1; Защита равна целочисленной сумме значения силы и живучести делённой на 3. После всех действия выводит в консоль Значения здоровья и брони.

1.4.7 def enter_room()

Данная функция выполняет логику перемещения игрока по комнатам. Функция изображена на Рисунке 7.

```
def enter_room(self):
    self.rooms_visited += 1
    if self.rooms_visited % self.settings["rooms_per_floor"] == 0:
        self.current_floor += 1
        print(f"\nВы перешли на {self.current_floor} этаж!")
        self.increase_monster_difficulty()

    junctions = [r for r in self.rooms if r.get("is_junction", False)]
    if not junctions:
        raise ValueError("Нет комнат с развилками!")

    total_chance = sum(r["chance"] for r in junctions)
    chosen = random.random() * total_chance
    selected_junction = None

    for room in junctions:
        chosen -= room["chance"]
        if chosen <= 0:
            selected_junction = room
            break

    self.current_room = selected_junction
    print(f"\n=== Вы подошли к развилке в: {self.current_room["name"]} ===")
    self.show_branches()
    direction = self.choose_direction()

    if direction == 1:
        next_room = self.current_room["left_branch"]
    elif direction == 2:
        next_room = self.current_room["right_branch"]
    else:
        print("Неверное направление.")
        return

    self.current_room = next_room
    print(f"\n=== Вы вошли в: {self.current_room["name"]} ===")
    self.handle_room_type(self.current_room["type"])
```

Рисунок 7 - Листинг кода для функции enter_room

Пояснение работы функции:

1) Сначала идёт расчёт посещённых комнат. Когда пользователь посещает последнюю комнату на этаже, при последующем выборе комнаты выводится сообщение о переходе на новый этаж и выполняется функция increase_monster_difficulty(пункт 1.4.10)

2) Затем проверяется наличие развилок с параметром is_junction равным True, если развилок не нашлось, то выводится исключение с сообщением

3) Случайным образом выбирается развилка из учёта их шансов на появление

4) Выполняется метод self.current_room, выводится сообщение о том какая развилка «выпала» пользователю, а затем применяются функции show_branches(пункт 1.4.8) choose_direction(пунтк 1.4.9), если возвращается 1, то переходит в левую ветку(пойти налево), если 2, то в правую ветку(пойти направо), иначе выводится в консоль сообщение о неверном направлении

5) После выводится сообщение о переходе в {название комнаты}

В самом конце выполняется функция handle_room_type(пункт 1.4.11).

1.4.8 def show_branches()

Данная функция отображает куда можно повернуть из развилки. Функция изображена на Рисунке 8.

```
def show_branches(self):
    left = self.current_room["left_branch"]
    right = self.current_room["right_branch"]

    if self.current_room["visibility"] == "unknown":
        print("Темнота не позволяет определить что находится впереди")
    else:
        print(f"1. Налево: {left["name"]} ({left["type"]})")
        print(f"2. Направо: {right["name"]} ({right["type"]})")
```

Рисунок 8 - Листинг кода для функции show_branches

Данная функция придаёт переменным left - левая ветка, right - правая ветка. Также выполняет условия, если развилка «unknown», то не отобразится что за комнаты, иначе показывается что слева и что справа.

1.4.9 def choose_direction()

Данная функция обрабатывает ввод пользователя для выбора направления. Функция изображена на Рисунке 9.

```
def choose_direction(self) -> int:
    while True:
        try:
            choice = input("Выберите направление (1 – налево, 2 – направо): ").strip()
            if choice in ["1", "2"]:
                return int(choice)
            else:
                print("Неверный выбор. Введите 1 или 2.")
        except ValueError:
            print("Введите число 1 или 2.")
```

Рисунок 9 - Листинг кода для функции choose_direction

Данная функция выполняется бесконечно циклом while True, пока не будет выполнен корректный ввод от пользователя. Сначала предоставляется выбор 1 - налево, 2 - направо, затем сравнивается с есть ли такой выбор в списке и возвращает его обратно, иначе выводит сообщение об ошибке.

1.4.10 def increase_monster_difficulty()

Данная функция выполняет процесс «усложнения» врагов, повышая их характеристики и уровень. Функция изображена на Рисунке 10.

```
def increase_monster_difficulty(self):
    multiplier = self.settings["monster_stat_per_floor"]

    for monster in self.monsters:
        monster["level"] = self.current_floor

        monster["health"] = math.ceil(monster["health"] * multiplier)
        monster["maxHealth"] = math.ceil(monster["maxHealth"] * multiplier)
        monster["attack"] = math.ceil(monster["attack"] * multiplier)
        monster["defense"] = math.ceil(monster["defense"] * multiplier)
        monster["health"] = min(monster["health"], monster["maxHealth"])
```

Рисунок 10 - Листинг кода для функции increase_monster_difficulty

Функция создаёт переменную multiplier и придаёт ей значение множителя из настроек игры, монстрам устанавливается уровень равный текущему этажу, затем для каждого монстра повышается одна характеристика поочереди на множитель с округлением до верхнего целого числа, после здоровье монстра устанавливается на максимальное, чтобы не вышло что здоровье больше максимального здоровья.

1.4.11 handle_room_type()

Данная функция исполняет логику комнаты через метод match-case. Функция изображена на Рисунке 11.

```
def handle_room_type(self, room_type: str):
    match room_type:
        case "combat":
            monster = self.encounter_monster()
            if monster:
                self.fight(monster)
            else:
                print("В комнате никого нет...")
        case "rest":
            self.rest()
        case "chest":
            self.open_chest()
        case _:
            print(f"Неизвестный тип комнаты: {room_type}")
```

Рисунок 11 - Листинг кода для функции handle_room_type

Методом match_case сравнивает тип комнаты с «выпавшей» игроку, если «combat», то создаёт переменную monster и выполняет функцию encounter_monster(пункт 1.4.17), а после функцию fight(пункт 1.4.18), иначе пишит что в комнате никого нет(шанс встречи не сработал). Тип комнаты «rest» выполняет функцию rest(пункт 1.4.12). Тип комнаты «chest» выполняет функцию open_chest(пункт 1.4.15). В случае если не совпадает с известными типами комнат, выводится сообщение об ошибке.

1.4.12 def rest()

Данная функция выполняет логику отдыха. Функция изображена на Рисунке 12.

```
def rest(self):
    self.player["health"] = self.player["max_health"]
    print("Вы отдохнули и полностью восстановили здоровье!")
    self.level_up()
```

Рисунок 12 - Листинг кода для функции rest

В данной функция производится отдых, здоровья игрока восстанавливается до максимального и выполняется функция level_up(пункт 1.4.13)

1.4.13 def level_up()

Данная функция сделана для проверки на получения нового уровня у игрока. Функция изображена на Рисунке 13.

```python
def level_up(self):
    while self.player["experience"] >= self.settings["exp_per_level"]:
        self.player["level"] += 1
        self.player["experience"] -= self.settings["exp_per_level"]
        stat_points = self.settings.get("stat_points_per_level", 2)
        self.player["stat_points"] += stat_points
        print(f"\nУровень повышен! Теперь у вас {self.player["level"]} уровень!")
        print(f"Вы получили {stat_points} очков характеристик для распределения.")
        self.distribute_stats(stat_points)
```

Рисунок 13 - Листинг кода для функции level_up

Здесь циклом прибавляем уровень игроку пока опыт превышает границу требуемого для повышения. Поднимаем уровень игрока на 1 и отнимает использованный опыт для повышения. Выдаём 2 очка навыков и выполняем функцию distribute_stats(пункт 1.4.14)

1.4.14 def distribute_stats()

Данная функция отображает меню повышения характеристик и исполняет это. Функция изображена на Рисунке 14.

```python
def distribute_stats(self, skill_points: int):
    print(f"\n=== Распределение очков навыков (всего {skill_points} очков) ===")
    print("1 - Живучесть")
    print("2 - Ловкость")
    print("3 - Сила")

    while skill_points > 0:
        print(f"\nОсталось распределить: {skill_points} очков.")
        choice = input("Выберите навык (1-3) или 'q' для выхода\n> ").strip()

        if choice == "q":
            print("Распределение прервано.")
            break

        if choice not in ["1", "2", "3"]:
            print("Ошибка: введите 1, 2, 3 или 'q'.")
            continue

        if choice == "1":
            self.player["stats"]["vitality"] += 1
            health_increase = 5
            self.player["max_health"] += health_increase
            self.player["health"] = min(self.player["health"] + health_increase, self.player["max_health"])
            print(f"+1 к Живучести! Максимальное здоровье увеличено на {health_increase}.")

        elif choice == "2":
            self.player["stats"]["agility"] += 1
            print("+1 к Ловкости!")

        elif choice == "3":
            self.player["stats"]["strength"] += 1
            damage_increase = 1
            self.player["damage"] += damage_increase
            print(f"+1   к Силе! Урон увеличен на {damage_increase}.")

        skill_points -= 1

    self.update_armor_from_defense()
    self.player["damage"] = (self.player["stats"]["strength"] +self.get_weapon_damage())
    self.player["max_health"] = self.player["stats"]["vitality"] * 5
    self.player["health"] = min(self.player["health"],self.player["max_health"])
```

Рисунок 14 - Листинг кода для функции level_up

Сначала отображаем какую характеристику хочет улучшить пользователь, после циклом задаём чтобы выполнялось пока очки характеристик больше 0. Командой input() спрашиваем у пользователя что улучшить, сравниваем ввод пользователя с ранее приведённым списком. За каждое очко живучести повышаем максимальное здоровье на 5, за каждое очко ловкости повышаем его на 1, за каждое очко силы повышаем его на 1, после выбора снимаем 1 очко характеристик. После обновляем характеристики для корректного отображения

1.4.15 def open_chest()

Данная функция выполняет логику получения добычи из сундука. Функция изображена на Рисунке 15.

```
def open_chest(self):
    loot_types = ["gold", "potion", "weapon", "armor"]
    loot = random.choice(loot_types)
    if loot == "gold":
        gold = random.randint(5, 20)
        self.player["gold"] += gold
        print(f"Вы нашли {gold} золота!")
    elif loot == "potion":
        potion_id = random.choice(list(self.potions.keys()))
        potion_name = self.potions[potion_id]["name"]
        self.add_item(potion_id, potion_name, "potion")
    elif loot == "weapon":
        weapon_id = random.choice(list(self.weapons.keys()))
        weapon_name = self.weapons[weapon_id]["name"]
        self.add_item(weapon_id, weapon_name, "weapon")
    elif loot == "armor":
        armor_id = random.choice(list(self.armors.keys()))
        armor_name = self.armors[armor_id]["name"]
        self.add_item(armor_id, armor_name, "armor")

    if len(self.player["inventory"]) > self.settings["max_inventory_slots"]:
        print(f"Инвентарь переполнен! Максимальное количество слотов: {self.settings["max_inventory_slots"]}")
        self.manage_inventory_overflow()
```

Рисунок 15 - Листинг кода для функции open_chest

Списком создаём возможный тип добычи, потом случайным образом выбираем что «выпадет» игроку и if-elif сравниваем. После успешного получения предметов проверяется не превышает ли текущее количество предметов максимальный размер инвентаря, если предметов больше чем разрешено, то выполняется функция manage_inventory_overflow(пункт 1.4.16)

1.4.16 def manage_inventory_overflow()

Данная функция выполняет процесс освобождения инвентаря при его переполнении. Функция изображена на Рисунке 16.

```
def manage_inventory_overflow(self):
    while len(self.player["inventory"]) > self.settings["max_inventory_slots"]:
        print("\nИнвентарь переполнен. Выберите предмет для удаления:")
        for i, item in enumerate(self.player["inventory"]):
            print(f"{i + 1}. {item["name"]} ({item["id"]})")
        try:
            choice = int(input("Номер предмета для удаления: ")) - 1
            if 0 <= choice < len(self.player["inventory"]):
                removed_item = self.player["inventory"].pop(choice)
                print(f"Удален предмет: {removed_item["name"]}")
            else:
                print("Неверный номер.")
        except ValueError:
            print("Введите число.")
```

Рисунок 16 - Листинг кода для функции manage_inventory_overflow

Здесь создаётся цикл, пока размер инвентаря больше максимально допустимого, то выводится сообщение о переполнении и требуют выбрать предмет для удаления. Перебирает все предметы в инвентаре с помощью enumerate(), присваивая каждому порядковый номер (i + 1). enumerate() добавляет индекс к элементам итерируемого объекта и возвращает их в виде пар. После запрашивается ввод пользователя какой предмет удалить. Идёт обработка ошибок, если введено число не в пределах инвентаря или введено не число.

1.4.17 def encounter_monster()

Данная функция находит монстра который может появиться в комнате и случайно выбирает его. Функция изображена на Рисунке 17.

```
def encounter_monster(self) -> Optional[Dict[str, Any]]:
    possible = [
        m for m in self.monsters
        if (m.get("location") == self.current_room["id"])
    ]
    return random.choice(possible) if possible else None
```

Рисунок 17 - Листинг кода для функции encounter_monster

Здесь создаётся переменная possible в которой создаётся список монстров которые подходят текущей комнате и случайно выбирает его, его такого не находит, то возвращает None.

1.4.18 def fight()

Данная функция реализует бой между игроком и монстром. Она содержит цикл боя, обработку действий игрока, расчёт урона, критических ударов, уклонения и прочего. Функция изображена в Приложении А. Победа засчитывается по флажку player_won(по умолчанию False). Игроку даётся 4 действия на выбор: атаковать, уклониться, использовать зелье и сбежать.

1) Игрок совершает удар. В игре есть параметр критического удара который рассчитывается во время боя. После этого ход монстра у него есть свой фиксированный шанс критического удара, а также шанс попадания, так как у игрока также имеется динамический шанс уклонения

2) Игрок может попытаться уклониться, никакой прибавки шанса к этому действию нет. При неудачном уклонении ход монстра.

3) Кнопка отправляет в меню выбора зелья из инвентаря, это действие не тратит ход

4) Игрок может попытаться сбежать от врага если тот окажется слишком силён, если не удалось то бьёт монстр, но шанс уклонения всё равно имеет, но равен половине от стандартного.

Выводится сообщение об ошибке если был неправильный ввод.

Если игрок погибает, то выводится сообщение о конце игры, а флажок game_over переходит в True. В случае гибели монстра флажок player_won переходит в True и выводится сообщение о победе и названии монстра, также добавляется опыт, золото, выполняется функция drop_items(пункт 1.4.18). В конце информативная строка о добытом опыте и золоте. В случае аварийного срабатывания смерти монстра выводится сообщение «Награда не выдана, бой не завершён победой».

1.4.19 def drop_items()

Данная функция выполняет логику выдачи награды с монстра. Функция изображена Рисунке 18.

```
def drop_items(self, monster: Dict[str, Any]):
    for item in monster.get("itemsDrop", []):
        if random.random() < item["chance"]:
            item_id = item["item"]
            item_data = None

            if item_id in self.weapons:
                item_data = self.weapons[item_id]
                item_type = "weapon"
            elif item_id in self.potions:
                item_data = self.potions[item_id]
                item_type = "potion"
            elif item_id in self.misc_items:
                item_data = self.misc_items[item_id]
                item_type = "misc"

            if item_data:
                self.add_item(item_id, item_data["name"], item_type, item_data.get("description", ""))

    if len(self.player["inventory"]) > self.settings["max_inventory_slots"]:
        print(f"Инвентарь переполнен! Максимальное количество слотов: {self.settings["max_inventory_slots"]}")
        self.manage_inventory_overflow()
```

Рисунок 18 - Листинг кода для функции drop_items

Эта функция реализует механику выпадения предметов из монстра после победы над ним. Она перебирает список возможных предметов, проверяет шанс выпадения, сравнивает какого типа выпали предметы и добавляет найденные предметы в инвентарь через функцию add_item(пункт 1.4.). Если инвентарь переполняется, то запускает функцию manage_inventory_overflow(пункт 1.4.16)

1.4.20 def use_potion()

Данная функция позволяет игроку использовать зелье из инвентаря для восстановления здоровья. Функция изображена на Рисунке 19.

```
def use_potion(self):
    potions = [
        item for item in self.player["inventory"]
        if item["id"] in self.potions and item["count"] > 0
    ]
    if not potions:
        print("Нет зелий в инвентаре!")
        return

    print("\n=== Доступные зелья ===")
    for i, item in enumerate(potions):
        potion_data = self.potions[item["id"]]
        heal = potion_data["heal"]
        print(f"{i + 1}. {potion_data["name"]} (+{heal} HP) ×{item["count"]}")


    try:
        choice = int(input("\nВведите номер зелья (0 — отмена): ")) - 1

        if choice == 0:
            print("Действие отменено.")
            return

        if choice < 0 or choice >= len(potions):
            print("Неверный номер! Попробуйте снова.")
            return

        selected_item = potions[choice]
        potion_data = self.potions[selected_item["id"]]
        heal_amount = potion_data["heal"]

        self.player["health"] = min(self.player["max_health"],self.player["health"] + heal_amount)

        print(f"Вы использовали {potion_data["name"]}!")
        print(f"Восстановлено: {heal_amount} HP")
        print(f"Здоровье: {self.player["health"]}/{self.player["max_health"]}")

        selected_item["count"] -= 1
        if selected_item["count"] <= 0:
            self.player["inventory"].remove(selected_item)
            print(" Зелье закончилось и удалено из инвентаря.")

    except ValueError:
        print("Ошибка: введите число!")
    except KeyError as e:
        print(f"Ошибка: не найдено зелье с ID {e}")
    except Exception as e:
        print(f"Неожиданная ошибка: {e}")
```

Рисунок 19 - Листинг кода для функции use_potion

Здесь идёт перебор доступных зелий в переменную potions из инвентаря, в случае их отсутствия отобразится «Нет зелий в инвентаре». Дальше выводит меню с выбором доступных зелий. Функция обрабатывает ошибки и имеет возможность отменить действие. При выпивании зелья используется функция min() чтобы здоровье не вышло за максимальное здоровье. Игроку отображается что использовали, сколько восстановили и текущее здоровья из максимума. Если после использования зелья оно закончилось(количество штук равно 0), то выводится соответствующее сообщение. Имеется обработка ошибок, если введено не число, ключ содержит ошибку в названии или код содержит иные ошибки.

1.4.21 def get_weapon_damage()

Данная функция определяет текущий урон оружия, экипированного игроком. Возвращает числовое значение урона. Функция изображена на Рисунке 20.

```
def get_weapon_damage(self) -> int:
    weapon_id = self.player["equipment"].get("weapon")
    if weapon_id and weapon_id in self.weapons:
        return self.weapons[weapon_id]["damage"]
    return 0
```

Рисунок 20 - Листинг кода для функции get_weapon_damage

Условием if проверяет наличие экипированного оружия и наличие данных об этом оружии из словаря.

1.4.22 def show_status()

Данная функция отображает статус персонажа. Функция изображена на Рисунке 21.

```
def show_status(self):
    print("\n=== Статус персонажа ===")
    print(f"Имя: {self.player["name"]}")
    print(f"Раса: {self.player["race"]}")
    print(f"Уровень: {self.player["level"]}")
    print(f"Опыт: {self.player["experience"]} / {self.settings["exp_per_level"]}")

    print(f"\nЗдоровье: {self.player["health"]} / {self.player["max_health"]}")
    print(f"Урон: {self.player["damage"]}")
    print(f"Броня: {self.player["armor"]}")

    dodge_chance = self.get_dodge_chance()
    dodge_percent = dodge_chance * 100
    print(f"Шанс уворота: {dodge_percent:.1f}%")

    crit_chance = self.get_crit_chance()
    crit_percent = crit_chance * 100
    print(f"Шанс критического удара: {crit_percent:.1f}%")

    print(f"\nХарактеристики:")
    print(f"  Сила: {self.player["stats"]["strength"]}")
    print(f"  Ловкость: {self.player["stats"]["agility"]}")
    print(f"  Живучесть: {self.player["stats"]["vitality"]}")
    print(f"  Вес: {self.player["stats"]["Weight"]}")
    print(f"  Рост: {self.player["stats"]["Height"]}")
```

Рисунок 21 - Листинг кода для функции show_status

В консоль выводится имя, раса, уровень, накопленный опыт и нужный для повышения уровня, максимальное и текущее здоровье, урон, броня, шанс уворота и критического удара в процентах. Основные характеристики, а именно сила, ловкость, живучесть, вес и рост.

1.4.23 def show_inventory()

Данная функция отображает содержимое инвентаря персонажа разбитое по категориям. Функция изображена в Приложении Б. В случае отсутствия вещей выводится сообщение «Инвентарь пуст.». Иначе выводится меню инвентаря с категориями предметов: зелья, оружие, броня, прочее. Кнопка «использовать зелье» появляется только если есть зелья для этого. Всегда выводятся кнопки «Выбросить предмет», «Посмотреть описание предмета» и «Вернуться в игру».

1.4.24 def add_item()

Данная функция выполняет логику добавления предмета в инвентарь. Функция изображена на Рисунке 22.

```
def add_item(self, item_id: str, name: str, item_type: str, description: str = ""):
    existing_item = next((i for i in self.player["inventory"] if i["id"] == item_id), None)
    if existing_item:
        existing_item["count"] += 1
    else:
        self.player["inventory"].append({
            "id": item_id,
            "name": name,
            "type": item_type,
            "count": 1,
            "description": description
        })
    print(f"Вы получили: {name}")
```

Рисунок 22 - Листинг кода для функции add_item

Функция добавляет предмет в инвентарь персонажа. Если предмет уже есть, то увеличивает его количество, иначе создать новую запись.

1.4.25 def equip_weapon()

Данная функция используется чтобы экипировать имеющиеся оружие. Функция изображена на Рисунке 23.

```
def equip_weapon(self):
    weapons = [
        i for i in self.player["inventory"]
        if i["id"] in self.weapons
    ]
    if not weapons:
        print("Нет оружия в инвентаре.")
        return

    if self.player["equipment"]["weapon"] != "None":
        print("Уже экипировано оружие. Сначала снимите текущее.")
        return

    print("\n=== Доступное оружие ===")
    for i, item in enumerate(weapons):
        weapon_data = self.weapons[item["id"]]
        print(f"{i+1}. {weapon_data['name']} (Урон: {weapon_data['damage']})")

    try:
        choice = int(input("Выберите оружие (номер, 0 — отмена): ")) - 1
        if choice == -1:
            print("Действие отменено.")
            return
        if 0 <= choice < len(weapons):
            selected_item = weapons[choice]
            self.player["inventory"].remove(selected_item)
            self.player["equipment"]["weapon"] = selected_item["id"]
            self.player["damage"] = (self.player["stats"]["strength"] + self.get_weapon_damage())
            print(f"Экипировано оружие: {self.weapons[selected_item['id']]['name']}")
        else:
            print("Неверный номер.")
    except ValueError:
        print("Введите число.")
```

Рисунок 23 - Листинг кода для функции equip_weapon

Здесь проверяется наличие оружия в инвентаре, свободность слота. Если есть оружие в инвентаре, то отображает его имя и урон. При выборе оружия экипирует в слот оружия, происходит перерасчёт урон персонажа и выводит надпись об экипированном оружии.

1.4.26 def equip_armor()

Действие аналогично с функций equip_weapon(пункт 1.4.25). Функция изображена на Рисунке 24.

```
def equip_armor(self):
    armors = [
        i for i in self.player["inventory"]
        if i["id"] in self.armors
    ]
    if not armors:
        print("Нет брони в инвентаре.")
        return

    if self.player["equipment"]["armor"] != "None":
        print("Уже экипирована броня. Сначала снимите текущую.")
        return

    print("\n=== Доступная броня ===")
    for i, item in enumerate(armors):
        armor_data = self.armors[item["id"]]
        print(f"{i+1}. {armor_data['name']} (Защита: {armor_data['defense']})")

    try:
        choice = int(input("Выберите броню (номер, 0 — отмена): ")) - 1
        if choice == -1:
            print("Действие отменено.")
            return
        if 0 <= choice < len(armors):
            selected_item = armors[choice]
            self.player["inventory"].remove(selected_item)
            self.player["equipment"]["armor"] = selected_item["id"]
            self.player["armor"] = self.armors[selected_item["id"]]["defense"]
            print(f"Экипирована броня: {self.armors[selected_item['id']]['name']}")
        else:
            print("Неверный номер.")
    except ValueError:
        print("Введите число.")
```

Рисунок 24 - Листинг кода для функции equip_armor

Данная функция проверяет наличие брони в инвентаре, свободность слота. При отсутствии брони в инвентаре выведется соответствующее сообщение. Выводится список доступной брони. После экипировки идёт перерасчёт брони.

1.4.27 def unequip()

Данная функция реализует снятие экипировки(оружия и брони) с персонажа. Функция изображена на Рисунке 25.

```
def unequip(self):
    print("\n=== Снятие экипировки ===")
    print("1. Снять оружие")
    print("2. Снять броню")
    choice = input("\n> ").strip()

    if choice == "1":
        weapon_id = self.player["equipment"]["weapon"]
        if weapon_id != "None":
            weapon_data = self.weapons[weapon_id]
            self.add_item(weapon_id, weapon_data["name"], "weapon")
            self.player["equipment"]["weapon"] = "None"
            self.player["damage"] = (self.player["stats"]["strength"] + self.get_weapon_damage())
            print(f"Вы сняли {weapon_data['name']}. Урон изменён.")
        else:
            print("Нет экипированного оружия")

    elif choice == "2":
        armor_id = self.player["equipment"]["armor"]
        if armor_id != "None":
            armor_data = self.armors[armor_id]
            self.add_item(armor_id, armor_data["name"], "armor")
            self.player["equipment"]["armor"] = "None"
            self.update_armor_from_defense()
            print(f"Вы сняли {armor_data['name']}. Защита изменена.")
        else:
            print("Нет экипированной брони")
    else:
        print("Неверный выбор! Используйте 1 или 2.")
```

Рисунок 25 - Листинг кода для функции unequip

Выводится меню снятия экипировки, где 1 - снять оружие, а 2 - снять броню. От пользователя получаем ввод, если 1, значит проверяет чтобы было надето оружие, затем добавляем его в инвентарь, снимает со слота экипировки, перерасчитываем урон и выводим что было снято, иначе сообщение об отсутствии экипированного оружия. Если 2, то будет аналогичный алгоритм, но для брони.

1.4.28 def save_game()

Данная функция сохраняет данные об текущей сессии в отдельный json файл. Функция изображена на Рисунке 26.

```
def save_game(self):
    save_data = {
        "player": self.player,
        "current_room": self.current_room,
        "current_floor": self.current_floor,
        "rooms_visited": self.rooms_visited,
        "game_over": self.game_over,
        "monsters": [
            {
                "id": m["id"],
                "name": m["name"],
                "level": m["level"],
                "health": m["health"],
                "maxHealth": m["maxHealth"],
                "attack": m["attack"],
                "defense": m["defense"],
                "experience": m["experience"],
                "goldDrop": m["goldDrop"],
                "itemsDrop": m["itemsDrop"],
                "location": m["location"]
            }
            for m in self.monsters
        ]
    }

    try:
        with open(os.path.join("data", "Saves.json"), "w", encoding="utf-8") as f:
            json.dump(save_data, f, ensure_ascii=False, indent=2)
        print("Игра сохранена в data/Saves.json")
    except Exception as e:
        print(f"Ошибка при сохранении: {e}")
```

Рисунок 26 - Листинг кода для функции save_game

Функция сохраняет все важные данные, а именно всю статистику персонажа, текущий этаж и комнату, посещённые комнаты, флажок game_over, данные о монстрах. Имеется обработка ошибок если что-то пойдёт не так при сохранении.

1.4.29 def load_game()

Данная функция загружает игру из последнего сохранения. Функция изображена на Рисунке 27.

```python
def load_game(self):
    save_path = os.path.join("data", "Saves.json")

    if not os.path.exists(save_path):
        print("Файл сохранения не найден: data/Saves.json")
        return

    try:
        with open(save_path, "r", encoding="utf-8") as f:
            save_data = json.load(f)

        self.player = save_data["player"]
        self.current_room = save_data["current_room"]
        self.current_floor = save_data["current_floor"]
        self.rooms_visited = save_data["rooms_visited"]
        self.game_over = save_data["game_over"]

        self.monsters = []
        for m_data in save_data["monsters"]:
            monster = {
                "id": m_data["id"],
                "name": m_data["name"],
                "level": m_data["level"],
                "health": m_data["health"],
                "maxHealth": m_data["maxHealth"],
                "attack": m_data["attack"],
                "defense": m_data["defense"],
                "experience": m_data["experience"],
                "goldDrop": m_data["goldDrop"],
                "itemsDrop": m_data["itemsDrop"],
                "location": m_data["location"]
            }
            self.monsters.append(monster)

        print("Игра загружена из data/Saves.json")
    except json.JSONDecodeError:
        print("Ошибка: файл сохранения повреждён (некорректный JSON).")
    except KeyError as e:
        print(f"Ошибка: отсутствует поле в сохранении: {e}")
    except Exception as e:
        print(f"Неожиданная ошибка при загрузке: {e}")
```

Рисунок 27 - Листинг кода для функции load_game

Проверяется наличие файла сохранения. Затем берёт данные из сохранения в текущую сессию. Для монстров создаётся пустой список и передаются словари с их данными.

1.4.30 def main_loop()

Функция является основным циклом игры. Она отображает меню выбора действий, обрабатывает ввод пользователя и вызывает соответствующие методы в зависимости от выбора. Работает до тех пор, пока не будет завершён игровой процесс. Функция изображена на Рисунке 28.

```
def main_loop(self):
    self.create_character()
    while not self.game_over:
        print("\n=== Меню выбора ===")
        print("1. Войти в комнату")
        print("2. Посмотреть статус персонажа")
        print("3. Посмотреть инвентарь")
        print("4. Экипировать оружие")
        print("5. Экипировать броню")
        print("6. Снять экипировку")
        print("7. Сохранить игру")
        print("8. Загрузить игру")
        print("9. Выйти из игры")

        try:
            choice = input("\n> ").strip()

            if choice == "1":
                self.enter_room()
            elif choice == "2":
                self.show_status()
            elif choice == "3":
                self.show_inventory()
            elif choice == "4":
                self.equip_weapon()
            elif choice == "5":
                self.equip_armor()
            elif choice == "6":
                self.unequip()
            elif choice == "7":
                self.save_game()
            elif choice == "8":
                self.load_game()
            elif choice == "9":
                print("Закрытие игры")
                break
            else:
                print("Неверное действие. Введите число от 1 до 9.")

        except KeyboardInterrupt:
            print("\nИгра прервана пользователем.")
            break
        except Exception as e:
            print(f"Неожиданная ошибка: {e}")

    if self.game_over:
        print("\n=== Игра окончена ===")
```

Рисунок 28 - Листинг кода для функции main_loop

Выход за диапазон действий или некорректный ввод выводит сообщение о соответствующей ошибке.

1.5 Точка входа

Для правильного логики запуска игры использует такую структуру: условием if проверяет чтобы файл запускался напрямую, создаёт новую переменную game и запускается класс Game и запускает цикл основной игры через main_loop().

Заключение

Основные результаты работы.

В ходе выполнения работы был разработан полностью функциональный прототип текстовой RPG-игры, соответствующий техническому заданию и включающий следующие ключевые компоненты:

1) Система создания персонажа с выбором расы и генерацией характеристик
2) Модульная структура игры на основе JSON-файлов с данными
3) Боевая система с реализацией механики уклонения, критических ударов
4) Система развития персонажа с прокачкой характеристик и распределением очков
5) Подземелье с процедурной генерацией комнат и развилок
6) Инвентарная система с возможностью экипировки предметов
7) Система сохранения прогресса игры

Достигнутые результаты.

Реализованы все базовые функциональные требования ТЗ:

1) Создание и развитие персонажа
2) Исследование подземелья
3) Боевая система
4) Инвентарь и экипировка
5) Сохранение/загрузка игры(Необязательное)

Были добавлены дополнительные функции:

1) Процедурная генерация контента
2) Система случайных событий
3) Более детальная система характеристик
4) Расширенная система предметов

Вывод.

Проведённая работа позволила создать работоспособный прототип RPG-игры, который может служить основой для дальнейшего развития проекта. Все поставленные задачи выполнены, технические требования соблюдены, функциональные возможности реализованы в полном объёме.

Приложение А

«обязательное»

```
def fight(self, monster: Dict[str, Any]):
    player_won = False
    print("\n--- Бой начался! ---")

    while self.player["health"] > 0 and monster["health"] > 0:
        print(f"\nВаше HP: {self.player["health"]} | Монстр HP: {monster["health"]}")
        action = input("1. Атаковать  2. Уклониться  3. Использовать зелье  4. Бежать\n> ").strip()

        crit_chance_monster = 0.05

        if action == "1":
            damage = self.player["damage"]
            is_crit = random.random() < self.get_crit_chance()
            if is_crit:
                damage *= 2
                print(f"Критический удар! Урон x2!")

            monster["health"] -= damage
            print(f"Вы нанесли {damage} урона!")

            if monster["health"] > 0:
                hit_chance = random.random()
                if hit_chance < self.get_dodge_chance():
                    print(f"Успешное уклонение!")
                else:
                    m_damage = max(1, monster["attack"] - self.player["armor"])
                    is_crit_monster = random.random() < crit_chance_monster
                    if is_crit_monster:
                        m_damage = int(m_damage * 1.5)
                        print(f"Критический удар монстра! Урон x1.5!")
                    self.player["health"] -= m_damage
                    print(f"Монстр нанес вам {m_damage} урона!")

        elif action == "2":
            if random.random() < self.get_dodge_chance():
                print("Вы успешно уклонились!")
            else:
                print("Уклонение не удалось!")
                m_damage = max(1, monster["attack"] - self.player["armor"])
                is_crit_monster = random.random() < crit_chance_monster
                if is_crit_monster:
                    m_damage = int(m_damage * 1.5)
                    print(f"Критический удар монстра при уклонении! Урон x1.5!")
                self.player["health"] -= m_damage
                print(f"Монстр нанес вам {m_damage} урона!")

        elif action == "3":
            self.use_potion()
            continue
```

Рисунок А.1

```
    elif action == "4":
        if random.random() < 0.5:
            print("Вы сбежали!")
            return
        else:
            print("Не удалось сбежать!")
            hit_chance = random.random()
            if hit_chance < self.get_dodge_chance() * 0.5:
                print(f"Удачливое уклонение! Шанс x0.5")
                pass
            m_damage = max(1, monster["attack"] - self.player["armor"])
            is_crit_monster = random.random() < crit_chance_monster
            if is_crit_monster:
                m_damage = int(m_damage * 1.5)
                print(f"Критический удар при побеге! Урон x1.5!")
            self.player["health"] -= m_damage
            print(f"Монстр ударил вас при побеге: {m_damage} урона!")

    else:
        print("Неверное действие.")

if self.player["health"] <= 0:
    print("Вы погибли... Игра окончена.")
    self.game_over = True
    exit()
elif monster["health"] <= 0:
    player_won = True
    monster["health"] = monster["maxHealth"]
    print(f"Победа! Вы победили {monster["name"]}!")

if player_won:
    self.player["experience"] += monster["experience"]
    self.player["gold"] += monster["goldDrop"]
    self.drop_items(monster)
    print(f"Вы получили: {monster["experience"]} опыта, {monster["goldDrop"]} золота.")
else:
    print("Награда не выдана — бой не завершён победой.")
```

Рисунок А.2

Приложение Б

«обязательное»

```
def show_inventory(self):
    if not self.player["inventory"]:
        print("Инвентарь пуст.")
        return

    print("\n=== Инвентарь ===")

    potions = []
    weapons = []
    armors = []
    misc_items = []

    for item in self.player["inventory"]:
        if item["id"] in self.potions:
            potions.append(item)
        elif item["id"] in self.weapons:
            weapons.append(item)
        elif item["id"] in self.armors:
            armors.append(item)
        else:
            misc_items.append(item)

    total_displayed = 0

    if potions:
        print("\nЗелья:")
        for i, item in enumerate(potions, 1):
            potion_data = self.potions[item["id"]]
            count = item["count"]
            heal = potion_data.get("heal", 0)
            print(f"{total_displayed + i}. {potion_data['name']} ×{count} (+{heal} HP)")
        total_displayed += len(potions)

    if weapons:
        print("\nОружие:")
        for i, item in enumerate(weapons, 1):
            weapon_data = self.weapons[item["id"]]
            count = item["count"]
            damage = weapon_data.get("damage", 0)
            print(f"{total_displayed + i}. {weapon_data['name']} ×{count} (урон: {damage})")
        total_displayed += len(weapons)

    if armors:
        print("\nБроня:")
        for i, item in enumerate(armors, 1):
            armor_data = self.armors[item["id"]]
            count = item["count"]
            defense = armor_data.get("defense", 0)
            print(f"{total_displayed + i}. {armor_data['name']} ×{count} (защита: {defense})")
        total_displayed += len(armors)

    if misc_items:
        print("\nПрочие предметы:")
        for i, item in enumerate(misc_items, 1):
            count = item["count"]
            name = item["name"]
            print(f"{total_displayed + i}. {name} ×{count}")
        total_displayed += len(misc_items)
```

Рисунок Б.1

```
print("\nВыберите действие:")
if potions:
    print("1. Использовать зелье")
if total_displayed > 0:
    print("2. Просмотреть описание предмета")
    print("3. Выбросить предмет")
print("4. Вернуться в игру")

choice = input("\nВаш выбор\n> ").strip()

if choice == "1" and potions:
    try:
        item_index = int(input(f"Введите номер зелья (1-{len(potions)}): ")) - 1
        if 0 <= item_index < len(potions):
            item = potions[item_index]
            potion_data = self.potions[item["id"]]
            heal_amount = potion_data["heal"]
            self.player["health"] = min(
                self.player["max_health"],
        self.player["health"] + heal_amount
    )
            item["count"] -= 1
            if item["count"] <= 0:
                self.player["inventory"].remove(item)
            print(f"Вы использовали {potion_data['name']}! +{heal_amount} HP. Здоровье: {self.player['health']}")
        else:
            print("Неверный номер зелья!")
    except ValueError:
        print("Введите число!")

elif choice == "2" and total_displayed > 0:
    try:
        item_num = int(input(f"Введите номер предмета (1-{total_displayed}): ")) - 1
        if item_num < 0 or item_num >= total_displayed:
            print("Неверный номер предмета!")
            return

        all_items = potions + weapons + armors + misc_items
        item = all_items[item_num]
        item_id = item["id"]

        item_data = None
        if item_id in self.potions:
            item_data = self.potions[item_id]
        elif item_id in self.weapons:
            item_data = self.weapons[item_id]
        elif item_id in self.armors:
            item_data = self.armors[item_id]
        else:
            item_data = {"description": f"{item['name']}. Детали неизвестны."}

        desc = item_data.get("description", "Описание отсутствует.")
        print(f"\nОписание: {desc}")

    except ValueError:
        print("Введите число!")
```

Рисунок Б.2

```
elif choice == "3" and total_displayed > 0:
    try:
        item_num = int(input(f"Введите номер предмета для выбрасывания (1-{total_displayed}): ")) - 1
        if item_num < 0 or item_num >= total_displayed:
            print("Неверный номер предмета!")
            return

        all_items = potions + weapons + armors + misc_items
        item = all_items[item_num]

        if item["count"] > 1:
            try:
                quantity = int(input(f"Сколько штук выбросить? (1-{item['count']}): "))
                if 1 <= quantity <= item["count"]:
                    item["count"] -= quantity
                    print(f"Вы выбросили {quantity} шт. {item['name']}")
                    if item["count"] == 0:
                        self.player["inventory"].remove(item)
                else:
                    print("Некорректное количество!")
            except ValueError:
                print("Введите число!")
        else:
            self.player["inventory"].remove(item)
            print(f"Вы выбросили {item['name']}")

    except ValueError:
        print("Введите число!")

elif choice == "4":
    pass
else:
    print("Некорректный выбор!")
```

Рисунок Б.3