

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

# ОТЧЕТ ПО РАЗРАБОТКЕ КОНСОЛЬНОЙ RPG ИГРЫ НА PYTHON

по дисциплине  
«Информатика и программирование»

Студент

гр. БИН-25-2

Ф.Р. Кучерчук

Ассистент

преподавателя

М.В. Водяницкий

## 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:

- создает 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 задает 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 дает 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 отдыха

### 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 (НР, 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)

```
1 Выберите
2 расу:
3 1 - Человек
4 2 - Эльф
5 3 - Дворф
6
7 > 2Ваш
8
9 персонаж создан!
10 HP: 85
11 ATK: 12
12 DEF: 6
13 AFI: 14Вы
14
15 входите в подземелье...Перед
16
17 вами развилка.
18 (1) Слева: ???
19 (2) Справа: Комната отдыхаКуда
20
21 пойти?
22 > 1
```

Рисунок 1 – Фрагмент создания персонажа

## 11. 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	Введение .....	3
1.1	Гейм-дизайн .....	3
2	Выполнение работы .....	4
2.1	Архитектура программы .....	4
2.2	Создание персонажа и класса .....	4
2.3	Система уровней и прокачки .....	4
2.4	Инвентарь и экипировка .....	5
2.5	Подземелье и генерация комнат .....	6
2.6	Боссы и прогрессия по этажам .....	7
2.7	Сложность .....	8
2.8	Боевая система .....	8
2.9	Сохранение данных .....	10
3	Тестирование .....	11
4	Заключение .....	12
	Приложение А .....	13

## 1 Введение

Разработка консольных текстовых игр представляет собой важный этап в обучении программированию, поскольку сочетает в себе работу с объектно-ориентированным проектированием, алгоритмами, пользовательским вводом и управлением состоянием. Такие проекты позволяют отработать навыки модульной архитектуры, обработки данных и взаимодействия с пользователем без необходимости использования графических библиотек.

Целью данной лабораторной работы является реализация прототипа текстовой RPG-игры, соответствующей техническому заданию, с акцентом на читаемость кода, логическую структуру и воспроизводимость игровых механик.

### 1.1 Гейм-дизайн

При проектировании игры были определены следующие ключевые принципы гейм-дизайна:

- Простота управления: игра полностью управляется через текстовое меню с числовым вводом, что обеспечивает доступность и предсказуемость.
- Прогрессия персонажа: игрок ощущает рост силы через повышение уровня, увеличение характеристик и получение более мощного снаряжения.
- Случайность и выбор: каждый запуск игры уникален благодаря процедурной генерации комнат, врагов и предметов; при этом игрок сохраняет контроль над решением — куда идти, кого атаковать, что экипировать.
- Баланс классов: три класса (Воин, Лучник, Маг) имеют разные стартовые параметры и стратегии развития, что поощряет повторные прохождения.

Эти принципы легли в основу архитектурных и программных решений, принятых при реализации.

## 2 Выполнение работы

### 2.1 Архитектура программы

Программа реализована в виде модульной структуры на языке Python. Основные компоненты:

- char\_classes.py — класс Hero, управление характеристиками, уровнем, опытом;
- enemy.py — класс Enemy с генерацией параметров;
- items\_classes.py — классы Item и прочие;
- dungeon.py — логика генерации подземелья и комнат;
- battle.py — функции боевой системы;
- main.py — основной цикл игры и взаимодействие с пользователем.

Такая структура позволяет легко расширять функционал, заменять отдельные модули и поддерживать читаемость кода. Каждый файл отвечает за одну зону ответственности, что соответствует принципам SOLID.

### 2.2 Создание персонажа и класса

При запуске игры пользователь выбирает Класс. На основе выбора генерируются характеристики в заданных диапазонах. Например:

- Воин: упор на живучесть;
- Лучник: дальний урон, выше ловкость;
- Маг: меньшая живучесть, но высокий урон.

```

1 class Hero(Entity):
2     def __init__(self, name):
3         super().__init__()
4         self.name = name.capitalize()
5         self.str = random.randint(10, 15)
6         self.dex = random.randint(10, 15)
7         self.int = random.randint(10, 15)
8         self.inventory = []
9         self.inventory_size = 10
10        self.exp = 0
11        self.exp_to_lvlup = ((1.15) ** self.lvl) * 100
12        self.gold = 0

```

Рисунок 2 – Фрагмент создания персонажа

Приведённый фрагмент демонстрирует инициализацию объекта героя с учётом выбранного класса. Случайная генерация характеристик происходит в рамках, заданных для каждого класса, что обеспечивает баланс и разнообразие игровых прохождений.



## 2.3 Система уровней и прокачки

Опыт начисляется за победы. При достижении порога опыта персонаж повышает уровень и получает очки характеристик, которые распределяются автоматически в зависимости от класса игрока.

```

1 def plus_exp(self, num):
2
3     self.exp += num
4     print(f"{num} опыта")
5     self.lvl_up()
6
7 def lvl_up(self):
8     while self.exp >= self.exp_to_lvlup:
9
10        self.exp -= self.exp_to_lvlup
11        self.lvl += 1
12        print("\n" + "="*30 + " НОВЫЙ УРОВЕНЬ! " + "="*30)
13        self.apply_stats_grow()
14        self.current_hp = self.max_hp
15        self.current_mana = self.max_mana
16        self.exp_to_lvlup = ((1.15) ** self.lvl) * 100
17        self.show_stats()
18        print("="*70)

```

Рисунок 3 – Фрагмент повышения уровня

Данный код реализует базовую логику повышения уровня: проверка накопленного опыта, увеличение уровня и вызов метода распределения очков. Это позволяет игроку ощущать прогресс по мере прохождения игры.

```

1 def apply_stats_grow(self):
2     self.str += 5
3     self.dex += 2
4     self.int += 1
5     self.max_hp = 120 + self.str * 3
6     self.current_hp = self.max_hp
7     self.max_mana = 20 + self.int
8     self.current_mana = self.max_mana
9     self.base_physical_dmg = 10 + self.str * 0.7
10    self.base_physical_res = 5 + self.str * 0.3
11    self.block_chance = 0.02 + 0.01 * self.lvl

```

Рисунок 4 – Фрагмент повышения уровня воина

В этом примере показано, как именно распределяются очки для класса «Воин»: приоритет отдаётся здоровью и атаке, что соответствует роли танка в бою. Аналогичные функции существуют для других классов.

## 2.4 Инвентарь и экипировка

Инвентарь реализован как список объектов. Экипировка (оружие и броня) находится в отдельных слотах и модифицирует базовые характеристики (например, меч даёт +5 к Attack).

```

1 class Weapon(Item):
2     def __init__(self, name, weapon_type, physical_dmg=0,
3         piercing_dmg=0, magic_dmg=0, **kwargs):
4         super().__init__(name, item_type="weapon", **kwargs)
5         self.weapon_type = weapon_type
6         self.physical_dmg = physical_dmg
7         self.piercing_dmg = piercing_dmg
8         self.magic_dmg = magic_dmg

```

Рисунок 5 – Фрагмент класса оружия

Класс `Weapon` наследуется от базового `Item` и содержит дополнительное поле `attack_bonus`, которое при экипировке суммируется с базовой атакой героя. Это упрощает расчёт урона в бою.

```

1 class Armor(Item):
2     def __init__(self, name, physical_res=0, magic_res=0, **
3         kwargs):
4         super().__init__(name, item_type="armor", **kwargs)
5         self.physical_res = physical_res
6         self.magic_res = magic_res

```

Рисунок 6 – Фрагмент класса брони

Аналогично, класс `Armor` добавляет бонус к защите. При смене экипировки старые бонусы корректно удаляются, а новые — применяются, что исключает ошибки в расчётах.

```

1 starter_staff = Weapon("Деревянный посох", "magic", magic_dmg
2     =14, lvl_required=1, clas_required="Mage")
3 starter_armor = Armor("Тканевая одежда", physical_res=2,
4     magic_res=1, lvl_required=1)

```

Рисунок 7 – Фрагмент создания предмета

Базовый класс `Item` обеспечивает единый интерфейс для всех предметов, включая зелья и ключи. Это упрощает работу с инвентарём и позволяет легко добавлять новые типы предметов.

## 2.5 Подземелье и генерация комнат

Подземелье строится динамически. После каждой комнаты игрок выбирает направление. Содержимое соседних комнат может быть скрыто («???») или раскрыто («Комната отдыха»), в зависимости от случайного фактора.

```

1 ROOM_TYPES = ["battle", "rest", "chest"]
2
3 class Dungeon:
4     def __init__(self):
5         self.floor = 1
6         self.room_count = 0
7         self.rooms_until_next_floor = 5 # После 5 комнат –
8         босс
9     def generate_room_pair(self):
10        left = random.choice(ROOM_TYPES)
11        right = random.choice(ROOM_TYPES)
12        return left, right

```

Рисунок 8 – Фрагмент генерации комнат

Функция генерации использует веса вероятностей для разных типов комнат, что позволяет контролировать частоту боёв, отдыха и сундуков. Также реализована логика видимости, создающая эффект неопределённости.

## 2.6 Боссы и прогрессия по этажам

После прохождения определённого количества комнат на этаже игрок попадает в комнату босса, который представляет усиленную версию монстра.

```

1 def create_boss(floor):
2     base_hp = 200 + floor * 50
3     base_dmg = 20 + floor * 5
4     bosses = [
5         ("Минотавр", 1.5, 1.3),
6         ("Лич", 1.2, 1.8),
7         ("Драконид", 1.7, 1.4),
8         ("Архидемон", 1.4, 1.6),
9     ]
10    name, hp_mult, dmg_mult = bosses[(floor - 1) % len(
11    bosses)]
12    hp = int(base_hp * hp_mult)
13    dmg = int(base_dmg * dmg_mult)
14    boss = Enemy(name, floor, hp, physical_dmg=dmg)
15    boss.evade_chance = 0.1 + floor * 0.02 # Боссы могут
    уклоняться
    return boss

```

Рисунок 9 – Фрагмент создания босса

Босс создаётся как экземпляр класса `Enemy`, но с увеличенными характеристиками и уникальным именем. Это добавляет разнообразие и повышает сложность на каждом новом этаже.

```

1 def start_boss_battle(hero, floor):
2     boss = create_boss(floor)
3     print(f"\n\n БОССБИТВА-! \n\n")
4     print(f"Этаж {floor}: {boss.name} yp(. {boss.lvl})
      выходит против вас!")
5     big_line()
6
7
8 if dungeon.advance_room():
9     print(f"\n{' '*50}")
10    print(f"\n Вы прошли все комнаты этажа {dungeon.
      floor}!")
11    print(f"Последний страж – БОСС! Готовьтесь к битве!
      ")
12    print(f"{' '*50}")
13
14    from logic.boss_battle import start_boss_battle
15    if not start_boss_battle(hero, dungeon.floor):
16        break # Герой погиб
17
18    # Переход на следующий этаж
19    dungeon.finish_floor()
20    print(f"\n\n Вы победили босса и спустились на этаж
      {dungeon.floor}!")

```

Рисунок 10 – Фрагмент вызова босса

Логика вызова босса активируется после счёта пройденных комнат. Игрок получает предупреждение, что впереди сильный противник, что даёт возможность подготовиться (использовать зелья, экипировать лучшее снаряжение).

## 2.7 Сложность

Сила врагов зависит от этажа на котором игрок с ним встречается.

```

1 def create_enemy(floor):
2     base_hp = 30 + floor * 8
3     base_dmg = 8 + floor * 3
4     templates = [
5         ("Гоблин", 1.0, 0.8),
6         ("Орк", 1.3, 1.0),
7         ("Тролль", 1.6, 1.2),
8         ("Демон", 1.9, 1.4),
9     ]
10    name, hp_mult, dmg_mult = random.choices(
11        templates,
12        weights=[4, 3, 2, 1],
13        k=1
14    )[0]
15    hp = int(base_hp * hp_mult)
16    dmg = int(base_dmg * dmg_mult)
17    return Enemy(name, floor, hp, physical_dmg=dmg)

```

Рисунок 11 – Фрагмент роста силы врагов

Коэффициент сложности линейно возрастает с каждым этажом. Это гарантирует, что игра остаётся вызовом даже на поздних стадиях, и побуждает игрока активно прокачивать персонажа.

## 2.8 Боевая система

Бой реализован как цикл «ход игрока → ход врага». Игрок может атаковать, открыть инвентарь или попытаться уклониться. Вероятность уклонения зависит от уровня, но увы растёт только у лучника.

```

1 def start_battle(hero, floor):
2     enemy = create_enemy(floor)
3     print(f"\n {enemy.name} ур(. {enemy.lv1}) атакует!")
4     big_line()
5
6     while not hero.is_dead and not enemy.is_dead:
7         print(f"\n{hero.name}: {hero.current_hp}/{hero.
8 max_hp} HP")
9         print(f"{enemy.name}: {enemy.current_hp}/{enemy.
10 max_hp} HP")
11         print("\n1. Атаковать")
12         print("2. Инвентарь")
13         print("3. Попытаться уклониться")
14         choice = input("> ").strip()
15
16         if choice == "1":
17             if hero.weapon and hero.weapon.weapon_type == "
18 melee":
19                 dmg = hero.physical_attack(enemy)
20             elif hero.weapon and hero.weapon.weapon_type ==
21 "ranged":
22                 dmg = hero.piercing_hit(enemy)
23             elif hero.weapon and hero.weapon.weapon_type ==
24 "magic":
25                 dmg = hero.magic_hit(enemy)
26             else:
27                 dmg = hero.physical_attack(enemy)
28                 print(f"Вы нанесли {dmg} урона.")
29
30         elif choice == "2":
31             hero.open_inventory()
32             continue # пропустить ход врага
33
34         elif choice == "3":
35             hero.evade_chance += 0.3
36             print("Вы готовитесь к уклонению!")
37         else:
38             print("Неизвестная команда.")
39             continue
40
41         # Сброс бонуса уклонения
42         hero.evade_chance = max(0, hero.evade_chance - 0.3)
43
44         if enemy.is_dead:
45             exp = 30 + enemy.lv1 * 20
46             gold = 15 + enemy.lv1 * 5
47             print(f"\n {enemy.name} повержен! +{exp} опыта,
48 +{gold} золота.")
49             hero.plus_exp(exp)
50             hero.add_to_inventory(Gold(gold))
51             return True
52
53         # Ход врага
54         dmg = enemy.physical_attack(hero)
55         print(f"{enemy.name} нанёс {dmg} урона.")
56
57     return not hero.is_dead

```

Рисунок 12 – Фрагмент боевой логики

Боевая система учитывает текущее состояние героя: здоровье, экипировку, наличие зелий. Также реализованы базовые механики уклонения и критического урона, зависящие от ловкости и удачи.

## 2.9 Сохранение данных

Состояние игры может быть сохранено в JSON-файл, включая:

- характеристики персонажа,
- инвентарь и экипировку,
- текущий этаж и прогресс.

Процесс сериализации преобразует объекты Python в словари, которые затем записываются в JSON. Это позволяет сохранять сложные структуры данных в человекочитаемом формате. При загрузке данные из JSON восстанавливаются в объекты соответствующих классов, что обеспечивает полное восстановление состояния игры без потерь.

Фрагменты кода функций сохранения и загрузки приведены в приложении А.

### 3 Тестирование

Для проверки корректности работы всех компонентов игры было проведено ручное функциональное тестирование. Основные проверяемые сценарии:

- Корректное создание персонажа с учётом выбранного класса и генерация характеристик в допустимых диапазонах.
- Правильное начисление опыта и повышение уровня после победы над врагом.
- Корректное применение эффектов экипировки: оружие увеличивает урон, броня — защиту.
- Генерация подземелья: все типы комнат (бой, отдых, сундук) появляются с ожидаемой частотой.
- Боевая система: урон зависит от атаки и защиты, уклонение работает только для Лучника и масштабируется с уровнем.
- Сохранение и загрузка: после перезапуска игры состояние персонажа восстанавливается без потерь.

Все протестированные сценарии завершились успешно. Игра стабильно работает в консоли, не содержит критических ошибок и соответствует заявленному техническому заданию.

## 4 Заключение

Разработан рабочий прототип консольной текстовой RPG, полностью соответствующий техническому заданию. Реализованы все ключевые механики: создание персонажа с расами, боевая система, прокачка, инвентарь, исследование подземелья и случайные события. Программа легко расширяема и соответствует требованиям к читаемости и модульности кода.



## Приложения

### Сохранение состояния игры

```

1 def save_game(hero, dungeon, slot_index):
2     """Сохраняет игру в указанный слот (0, 1, 2)."""
3     if not (0 <= slot_index < 3):
4         print("❌ Неверный слот.")
5         return False
6
7     filename = SAVE_SLOTS[slot_index]
8
9     # Данные героя
10    hero_data = {
11        "name": hero.name,
12        "class": hero.__class__.__name__,
13        "lvl": hero.lvl,
14        "exp": hero.exp,
15        "exp_to_lvlup": hero.exp_to_lvlup,
16        "current_hp": hero.current_hp,
17        "max_hp": hero.max_hp,
18        "current_mana": hero.current_mana,
19        "max_mana": hero.max_mana,
20        "str": hero.str,
21        "dex": hero.dex,
22        "int": hero.int,
23        "gold": hero.gold,
24        "weapon_name": hero.weapon.name if hero.weapon else
25        None,
26        "armor_name": hero.armor.name if hero.armor else
27        None,
28        "inventory_names": [item.name for item in hero.
29        inventory if item.item_type != "gold"]
30    }
31
32    # Данные подземелья
33    dungeon_data = {
34        "floor": dungeon.floor,
35        "room_count": dungeon.room_count,
36        "rooms_until_next_floor": dungeon.
37        rooms_until_next_floor
38    }
39
40    save_data = {
41        "hero": hero_data,
42        "dungeon": dungeon_data
43    }
44
45    try:
46        with open(filename, "w", encoding="utf-8") as f:
47            json.dump(save_data, f, ensure_ascii=False,
48            indent=2)
49        print(f"✅ Игра сохранена в слот {slot_index + 1}!")
50        return True
51    except Exception as e:
52        print(f"❌ Ошибка сохранения: {e}")
53        return False

```

Рисунок 13 – Функция сохранения

## Загрузка состояния игры

```

1 def load_game(slot_index):
2     """Загружает игру из указанного слота."""
3     if not (0 <= slot_index < 3):
4         print("❌ Неверный слот.")
5         return None, None
6
7     filename = SAVE_SLOTS[slot_index]
8     if not os.path.exists(filename):
9         print("❌ Слот пуст.")
10        return None, None
11
12    try:
13        with open(filename, "r", encoding="utf-8") as f:
14            data = json.load(f)
15
16            hero_data = data["hero"]
17            dungeon_data = data["dungeon"]
18
19            # Восстановление героя
20            name = hero_data["name"]
21            cls_name = hero_data["class"]
22            cls = {"Melee": Melee, "Ranger": Ranger, "Mage":
23            Mage}[cls_name]
24            hero = cls(name)
25
26            hero.lvl = hero_data["lvl"]
27            hero.exp = hero_data["exp"]
28            hero.exp_to_lvlup = hero_data["exp_to_lvlup"]
29            hero.current_hp = hero_data["current_hp"]
30            hero.max_hp = hero_data["max_hp"]
31            hero.current_mana = hero_data["current_mana"]
32            hero.max_mana = hero_data["max_mana"]
33            hero.str = hero_data["str"]
34            hero.dex = hero_data["dex"]
35            hero.int = hero_data["int"]
36            hero.gold = hero_data["gold"]
37
38            weapon_name = hero_data.get("weapon_name")
39            armor_name = hero_data.get("armor_name")
40            hero.weapon = ITEM_MAP.get(weapon_name) if
41            weapon_name else None
42            hero.armor = ITEM_MAP.get(armor_name) if armor_name
43            else None
44
45            inventory_names = hero_data.get("inventory_names",
46            [])
47            hero.inventory = [ITEM_MAP[name] for name in
48            inventory_names if name in ITEM_MAP]
49
50            # Восстановление подземелья
51            from labs.rpg.game.dungeon import Dungeon
52            dungeon = Dungeon()
53            dungeon.floor = dungeon_data["floor"]
54            dungeon.room_count = dungeon_data["room_count"]
55            dungeon.rooms_until_next_floor = dungeon_data["
56            rooms_until_next_floor"]
57
58            print(f"❌ Игра загружена из слота {slot_index + 1}!")
59            return hero, dungeon
60
61    except Exception as e:
62        print(f"❌ Ошибка загрузки: {e}")
63        return None, None

```

Рисунок 14 – Функция загрузки