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

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:

- 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 (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

> 2Ваш

 персонаж создан!
HP: 85
ATK: 12
DEF: 6
AFI: 14Вы

 входите в подземелье...Перед

 вами развилка.
(1) Слева: ???
(2) Справа: Комната отдыхаКуда

 пойти?
> 1

```

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

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.

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.

```
class Hero(Entity):
    def __init__(self, name):
        super().__init__()
        self.name = name.capitalize()
        self.str = random.randint(10, 15)
        self.dex = random.randint(10, 15)
        self.int = random.randint(10, 15)
        self.inventory = []
        self.inventory_size = 10
        self.exp = 0
        self.exp_to_lvlup = ((1.15) ** self.lvl) * 100
        self.gold = 0
```

Рисунок 1 – Фрагмент создания персонажа

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.

```
def plus_exp(self, num):

    self.exp += num
    print(f"+{num} опыта")
    self.lvl_up()

def lvl_up(self):
    while self.exp >= self.exp_to_lvlup:

        self.exp -= self.exp_to_lvlup
        self.lvl += 1
        print("\n" + "="*30 + " НОВЫЙ УРОВЕНЬ! " + "="*30)
        self.apply_stats_grow()
        self.current_hp = self.max_hp
        self.current_mana = self.max_mana
        self.exp_to_lvlup = ((1.15) ** self.lvl) * 100
        self.show_stats()
        print("="*70)
```

Рисунок 2 – Фрагмент повышения уровня

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

Рисунок 3 – Фрагмент повышения уровня воина

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).

```
class Weapon(Item):
    def __init__(self, name, weapon_type, physical_dmg=0,
    piercing_dmg=0, magic_dmg=0, **kwargs):
        super().__init__(name, item_type="weapon", **kwargs)
        self.weapon_type = weapon_type
        self.physical_dmg = physical_dmg
        self.piercing_dmg = piercing_dmg
        self.magic_dmg = magic_dmg
```

Рисунок 4 – Фрагмент класса оружия

```
class Armor(Item):
    def __init__(self, name, physical_res=0, magic_res=0, **
    kwargs):
        super().__init__(name, item_type="armor", **kwargs)
        self.physical_res = physical_res
        self.magic_res = magic_res
```

Рисунок 5 – Фрагмент класса брони

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

Рисунок 6 – Фрагмент создания предмета

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.

```
ROOM_TYPES = ["battle", "rest", "chest"]

class Dungeon:
    def __init__(self):
        self.floor = 1
        self.room_count = 0
        self.rooms_until_next_floor = 5  # После 5 комнат — босс

    def generate_room_pair(self):
        left = random.choice(ROOM_TYPES)
        right = random.choice(ROOM_TYPES)
        return left, right
```

Рисунок 7 – Фрагмент генерации комнат

2.6 Подземелье и генерация комнат

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

```
def create_boss(floor):
    base_hp = 200 + floor * 50
    base_dmg = 20 + floor * 5
    bosses = [
        ("Минотавр", 1.5, 1.3),
        ("Лич", 1.2, 1.8),
        ("Драконид", 1.7, 1.4),
        ("Архидемон", 1.4, 1.6),
    ]
    name, hp_mult, dmg_mult = bosses[(floor - 1) % len(bosses)]
    hp = int(base_hp * hp_mult)
    dmg = int(base_dmg * dmg_mult)
    boss = Enemy(name, floor, hp, physical_dmg=dmg)
    boss.evade_chance = 0.1 + floor * 0.02  # Боссы могут уклоняться
    return boss
```

Рисунок 8 – Фрагмент создания босса

```
def start_boss_battle(hero, floor):
    boss = create_boss(floor)
    print(f"\n БОССБИТВА-! ")
    print(f"Этаж {floor}: {boss.name} ур(. {boss.lvl}) выходит против вас!")
    big_line()


if dungeon.advance_room():
            print(f"\n{'='*50}")
            print(f" Вы прошли все комнаты этажа {dungeon.floor}!")
            print(f"Последний страж — БОСС! Готовьтесь к битве!")
            print(f"{'='*50}")

            from logic.boss_battle import start_boss_battle
            if not start_boss_battle(hero, dungeon.floor):
                break  # Герой погиб

            # Переход на следующий этаж
            dungeon.finish_floor()
            print(f"\n Вы победили босса и спустились на этаж {dungeon.floor}!")
```

Рисунок 9 – Фрагмент вызова босса

2.7 Сложность

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

```
def create_enemy(floor):
    base_hp = 30 + floor * 8
    base_dmg = 8 + floor * 3
    templates = [
        ("Гоблин", 1.0, 0.8),
        ("Орк", 1.3, 1.0),
        ("Тролль", 1.6, 1.2),
        ("Демон", 1.9, 1.4),
    ]
    name, hp_mult, dmg_mult = random.choices(
        templates,
        weights=[4, 3, 2, 1],
        k=1
    )[0]
    hp = int(base_hp * hp_mult)
    dmg = int(base_dmg * dmg_mult)
    return Enemy(name, floor, hp, physical_dmg=dmg)
```

Рисунок 10 – Фрагмент роста силы врагов

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.

```python
def start_battle(hero, floor):
    enemy = create_enemy(floor)
    print(f"\n  {enemy.name} ур(. {enemy.lvl}) атакует!")
    big_line()

    while not hero.is_dead and not enemy.is_dead:
        print(f"\n{hero.name}: {hero.current_hp}/{hero.max_hp} HP")
        print(f"{enemy.name}: {enemy.current_hp}/{enemy.max_hp} HP")
        print("\n1. Атаковать")
        print("2. Инвентарь")
        print("3. Попытаться уклониться")
        choice = input("> ").strip()

        if choice == "1":
            if hero.weapon and hero.weapon.weapon_type == "melee":
                dmg = hero.physical_attack(enemy)
            elif hero.weapon and hero.weapon.weapon_type == "ranged":
                dmg = hero.piercing_hit(enemy)
            elif hero.weapon and hero.weapon.weapon_type == "magic":
                dmg = hero.magic_hit(enemy)
            else:
                dmg = hero.physical_attack(enemy)
            print(f"Вы нанесли {dmg} урона.")

        elif choice == "2":
            hero.open_inventory()
            continue  # пропустить ход врага

        elif choice == "3":
            hero.evade_chance += 0.3
            print("Вы готовитесь к уклонению!")
        else:
            print("Неизвестная команда.")
            continue

        # Сброс бонуса уклонения
        hero.evade_chance = max(0, hero.evade_chance - 0.3)

        if enemy.is_dead:
            exp = 30 + enemy.lvl * 20
            gold = 15 + enemy.lvl * 5
            print(f"\n {enemy.name} повержен! +{exp} опыта, +{gold} золота.")
            hero.plus_exp(exp)
            hero.add_to_inventory(Gold(gold))
            return True

        # Ход врага
        dmg = enemy.physical_attack(hero)
        print(f"{enemy.name} нанёс {dmg} урона.")

    return not hero.is_dead
```

Рисунок 11 – Фрагмент боевой логики

2.9 Сохранение данных

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

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

```
def save_game(hero, dungeon, slot_index):
    """Сохраняет игру в указанный слот (0, 1, 2)."""
    if not (0 <= slot_index < 3):
        print("❌ Неверный слот.")
        return False

    filename = SAVE_SLOTS[slot_index]

    # Данные героя
    hero_data = {
        "name": hero.name,
        "class": hero.__class__.__name__,
        "lvl": hero.lvl,
        "exp": hero.exp,
        "exp_to_lvlup": hero.exp_to_lvlup,
        "current_hp": hero.current_hp,
        "max_hp": hero.max_hp,
        "current_mana": hero.current_mana,
        "max_mana": hero.max_mana,
        "str": hero.str,
        "dex": hero.dex,
        "int": hero.int,
        "gold": hero.gold,
        "weapon_name": hero.weapon.name if hero.weapon else None,
        "armor_name": hero.armor.name if hero.armor else None,
        "inventory_names": [item.name for item in hero.inventory if item.item_type != "gold"]
    }

    # Данные подземелья
    dungeon_data = {
        "floor": dungeon.floor,
        "room_count": dungeon.room_count,
        "rooms_until_next_floor": dungeon.rooms_until_next_floor
    }

    save_data = {
        "hero": hero_data,
        "dungeon": dungeon_data
    }

    try:
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(save_data, f, ensure_ascii=False, indent=2)
        print(f"✅ Игра сохранена в слот {slot_index + 1}!")
        return True
    except Exception as e:
        print(f"❌ Ошибка сохранения: {e}")
        return False
```

Рисунок 12 – Фрагмент сохранения

```python
def load_game(slot_index):
    """Загружает игру из указанного слота."""
    if not (0 <= slot_index < 3):
        print("❌ Неверный слот.")
        return None, None

    filename = SAVE_SLOTS[slot_index]
    if not os.path.exists(filename):
        print("❌ Слот пуст.")
        return None, None

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

        hero_data = data["hero"]
        dungeon_data = data["dungeon"]

        # Восстановление героя
        name = hero_data["name"]
        cls_name = hero_data["class"]
        cls = {"Melee": Melee, "Ranger": Ranger, "Mage": Mage}[cls_name]
        hero = cls(name)

        hero.lvl = hero_data["lvl"]
        hero.exp = hero_data["exp"]
        hero.exp_to_lvlup = hero_data["exp_to_lvlup"]
        hero.current_hp = hero_data["current_hp"]
        hero.max_hp = hero_data["max_hp"]
        hero.current_mana = hero_data["current_mana"]
        hero.max_mana = hero_data["max_mana"]
        hero.str = hero_data["str"]
        hero.dex = hero_data["dex"]
        hero.int = hero_data["int"]
        hero.gold = hero_data["gold"]

        weapon_name = hero_data.get("weapon_name")
        armor_name = hero_data.get("armor_name")
        hero.weapon = ITEM_MAP.get(weapon_name) if weapon_name else None
        hero.armor = ITEM_MAP.get(armor_name) if armor_name else None

        inventory_names = hero_data.get("inventory_names", [])
        hero.inventory = [ITEM_MAP[name] for name in inventory_names if name in ITEM_MAP]

        # Восстановление подземелья
        from labs.rpg.game.dungeon import Dungeon
        dungeon = Dungeon()
        dungeon.floor = dungeon_data["floor"]
        dungeon.room_count = dungeon_data["room_count"]
        dungeon.rooms_until_next_floor = dungeon_data["rooms_until_next_floor"]

        print(f"✅ Игра загружена из слота {slot_index + 1}!")
        return hero, dungeon

    except Exception as e:
        print(f"❌ Ошибка загрузки: {e}")
        return None, None
```

Рисунок 13 – Фрагмент загрузки

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.