

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 ЛАБОРАТОРНОЙ РАБОТЕ №8
по дисциплине
«Информатика и программирование»

Студент

гр. БИН-25-3

Герцов Д.Е.

Ассистент

преподавателя

М.В. Водяницкий

Содержание

1	Введение	3
2	Выполнение работы	4
2.1	Класс Character – инициализация и свойства	4
2.2	Класс Character – методы взаимодействия	4
2.3	Инвентарь и экипировка	6
2.4	Враги – базовые классы и первые типы	7
2.5	Враги – продвинутые типы и генерация	7
2.6	Расы – ядро уникальных механик	8
2.7	Создание персонажа	10
2.8	Основной игровой цикл	10
2.9	Боевая система и механики	12
3	Заключение	14

Задание

Реализовать консольную текстовую RPG–игру, демонстрирующую основные игровые механики:

- Создание персонажа с выбором расы и случайными характеристиками
- Боевая система с уклонением, эффектами и уникальными способностями
- Инвентарь и экипировка
- Прокачка и распределение очков характеристик
- Исследование подземелья с развилками и случайными событиями

1 Введение

Развитие игровой индустрии на заре её становления было неразрывно связано с созданием текстовых RPG, которые демонстрировали возможности программирования для имитации сложных игровых миров.

Такие игры, запускаемые в консоли, требовали от разработчика глубокого понимания объектно-ориентированного программирования, работы с данными и проектирования игровых механик.

Целью данной лабораторной работы является реализация прототипа консольной текстовой RPG-игры, отражающего ключевые аспекты игрового процесса: создание персонажа с уникальными характеристиками, боевую систему с элементами случайности, управление инвентарём, прокачку и исследование процедурно генерируемого подземелья.

В ходе выполнения работы применяются принципы ООП, работа со случайными величинами, обработка пользовательского ввода и организация сложной логики взаимодействия между игровыми сущностями.

Актуальность работы обусловлена необходимостью закрепления навыков программирования на языке Python, а также понимания архитектурных решений, лежащих в основе даже самых простых игровых проектов.

Реализованный прототип может служить основой для дальнейшего расширения функционала и изучения более сложных игровых движков.

2 Выполнение работы

2.1 Класс Character – инициализация и свойства

Реализован базовый класс Character с инициализацией характеристик и вычисляемыми свойствами: ИМТ, урон, защита, уклонение.

На рисунке 1 представлен код.

```

1 import random
2
3 class Character:
4     def __init__(self, name, race, health, damage, defense, agility, height, weight):
5         self.name = name
6         self.race = race
7         self.health = health
8         self.max_health = health
9         self.damage = damage
10        self.defense = defense
11        self.agility = agility
12        self.height = height
13        self.weight = weight
14
15        self.level = 1
16        self.exp = 0
17        self.stat_points = 0
18
19        self.weapon_name = "Палка"
20        self.weapon_attack = 0
21        self.armor_name = "Бандаж"
22        self.armor_defense = 0
23
24        self.effects = []
25
26    @property
27    def index_mass(self):
28        height_m = self.height / 100
29        return self.weight / (height_m ** 2)
30
31    @property
32    def total_attack(self):
33        return self.damage + self.weapon_attack
34
35    @property
36    def total_defense(self):
37        imt = self.index_mass
38        bonus_defense = 0
39        if imt < 18.5:
40            bonus_defense = -3
41        elif imt > 25:
42            bonus_defense = 5
43        return self.defense + self.armor_defense + bonus_defense
44
45    @property
46    def evasion_chance(self):
47        base_dodge = self.agility / 100
48        imt = self.index_mass
49        bonus_dodge = 0
50        if imt < 18.5:
51            bonus_dodge = 0.2
52        elif 18.5 <= imt <= 25:
53            bonus_dodge = 0.05
54        else:
55            bonus_dodge = -0.15
56        total = base_dodge + bonus_dodge
57        return max(0.0, min(0.7, total))

```

Рисунок 1 – Листинг character1.py

- 1) Определены основные характеристики
- 2) Реализованы свойства `index_mass`, `total_attack`, `evasion_chance`

Зачем это нужно: Базовый класс Character является фундаментом всей игровой системы. Он обеспечивает единый интерфейс для всех сущностей (игрока и врагов), что упрощает расширение и поддержку кода. Свойства, такие как `evasion_chance`, позволяют реализовать сложную боевую механику без дублирования логики.

2.2 Класс Character – методы взаимодействия

Реализованы методы взаимодействия: получение урона, атака, лечение, прокачка, обработка эффектов.

На рисунке 2 – код.

```

1  def apply_effect(self, effect_name, duration, damage_per_turn=0):
2      self.effects.append({
3          "name": effect_name,
4          "duration": duration,
5          "damage_per_turn": damage_per_turn
6      })
7
8  def update_effects(self):
9      new_effects = []
10     for effect in self.effects:
11         if effect["duration"] > 0:
12             if effect["damage_per_turn"] > 0:
13                 print(f" → {self.name} получает {effect['damage_per_turn']} урона от '{effect['name']}'")
14                 self.take_damage(effect["damage_per_turn"])
15                 effect["duration"] -= 1
16                 new_effects.append(effect)
17             else:
18                 print(f" → Эффект '{effect['name']}' закончился.")
19     self.effects = new_effects
20
21 def take_damage(self, damage):
22     if random.random() < self.evasion_chance:
23         print(f" → {self.name} уклонился!")
24         return 0
25
26     actual_damage = max(1, damage - self.total_defense)
27     self.health -= actual_damage
28     if self.health < 0:
29         self.health = 0
30     print(f" → {self.name} получил {actual_damage} урона. Здоровье: {self.health}")
31     return actual_damage
32
33 def attack(self, other):
34     print(f"{self.name} бьет {other.name}!")
35     other.take_damage(self.total_attack)
36
37 def is_alive(self):
38     return self.health > 0
39
40 def heal(self, amount):
41     old_health = self.health
42     self.health = min(self.max_health, self.health + amount)
43     healed = self.health - old_health
44     print(f" → {self.name} восстановил {healed} HP. Теперь HP: {self.health}")
45
46 def gain_exp(self, amount):
47     self.exp += amount
48     needed = self.level * 50
49     if self.exp >= needed:
50         self.level_up()
51
52 def level_up(self):
53     self.level += 1
54     self.stat_points += 3
55     print(f"\nУровень повышен! Теперь у вас {self.level}- уровень. Получено 3 очка характеристик.")
56     self.apply_statpoints()
57
58 def apply_statpoints(self):
59     while self.stat_points > 0:
60         print(f"\nУ вас {self.stat_points} очкоав(/). Куда вложите?")
61         print("1. +5 Здоровья")
62         print("2. +1 Защиты")
63         print("3. +1 Атаки")
64         print("4. +1 Ловкости")
65
66         choice = input("Выбор (1-4): ").strip()
67         if choice == "1":
68             self.max_health += 5
69             self.health += 5
70             self.stat_points -= 1
71         elif choice == "2":
72             self.defense += 1
73             self.stat_points -= 1
74         elif choice == "3":
75             self.damage += 1
76             self.stat_points -= 1
77         elif choice == "4":
78             self.agility += 1
79             self.stat_points -= 1
80         else:
81             print("Цифру от 1 до 4, мудила!")

```

Рисунок 2 – Листинг character2.py

- 1) Методы `take_damage`, `attack`, `heal`
- 2) Система эффектов через `update_effects`
- 3) Прокачка через `apply_statpoints`

Зачем это нужно: Методы взаимодействия (`take_damage`, `attack`) реализуют основную игровую логику боя. Система эффектов позволяет добавлять временные состояния (отравление, кровотечение), что значительно расширяет тактическую глубину игры. Прокачка через `apply_statpoints` даёт игроку контроль над развитием персонажа.

2.3 Инвентарь и экипировка

Реализован класс `Inventory` для управления предметами.

Поддерживается ограничение вместимости, экипировка оружия и брони, использование зелий.

На рисунке 3 — код.

```

1 class Inventory:
2     def __init__(self, capacity=10):
3         self.items = []
4         self.capacity = capacity
5         self.gold = 0
6
7     def add_item(self, item):
8         if len(self.items) >= self.capacity:
9             print("Сори инвентарь полон, попробуй чтото- выкинуть:")
10            self.show()
11            idx = input("Номер предмета для выброса ничего(0-): ")
12            if idx.isdigit() and 1 <= int(idx) <= len(self.items):
13                self.items.pop(int(idx)-1)
14            else:
15                print("Предмет не добавлен")
16                return False
17            self.items.append(item)
18            print(f"Получен предмет: {item.get('name', '???')}")
19            return True
20
21    def show(self):
22        print("\nИнвентарь-----")
23        print(f"Золото: {self.gold}")
24        if not self.items:
25            print("Пусто.")
26        else:
27            for i, item in enumerate(self.items, 1):
28                print(f"{i}. {item['name']}")
29
30    def use_item(self, index, character):
31        if 1 <= index <= len(self.items):
32            item = self.items[index - 1]
33            if item["type"] == "potion":
34                character.heal(item["heal"])
35                self.items.pop(index - 1)
36            elif item["type"] == "weapon":
37                old_name, old_atk = character.weapon_name, character.weapon_attack
38                character.weapon_name = item["name"]
39                character.weapon_attack = item["attack"]
40                self.items.pop(index - 1)
41                if old_name != "Палка":
42                    self.items.append({"name": old_name, "type": "weapon", "attack": old_atk})
43                print(f"Экипировано: {item['name']}")
44            elif item["type"] == "armor":
45                old_name, old_def = character.armor_name, character.armor_defense
46                character.armor_name = item["name"]
47                character.armor_defense = item["defense"]
48                self.items.pop(index - 1)
49                if old_name != "Бандаж":
50                    self.items.append({"name": old_name, "type": "armor", "defense": old_def})
51                print(f"Экипировано: {item['name']}")
52            else:
53                print("Нельзя использовать.")
54        else:
55            print("Нет такого предмета!")

```

Рисунок 3 – Листинг `inventory.py`

Зачем это нужно: Класс `Inventory` управляет ресурсами игрока, что является ключевым элементом RPG-механик. Ограничение вместимости создаёт стратегический выбор: что взять, а что оставить. Экипировка напрямую влияет на боевые характеристики, связывая прогресс в исследовании с боевой мощностью.

2.4 Враги – базовые классы и первые типы

Создан базовый класс `Enemy` и два уникальных врага: Пингвин (кровотечение) и Человек в маске (воскрешение).

На рисунке 4 – реализация.

```

1 import random
2
3 # === Враги – часть 1 ===
4
5 class Enemy(Character):
6     def __init__(self, name, hp, attack, defense, agility, height=170, weight=60):
7         super().__init__(name, "Монстр", hp, attack, defense, agility, height, weight)
8         self.exp_reward = 0
9
10    def set_exp(self, exp):
11        self.exp_reward = exp
12
13    def special_ability(self, target):
14        pass
15
16    def attack(self, other):
17        print(f"{self.name} бьёт {other.name}!")
18        other.take_damage(self.total_attack)
19        if random.random() < 0.3:
20            self.special_ability(other)
21
22
23 class PingVin735(Enemy):
24     def __init__(self):
25         super().__init__("Пингвин", 25, 6, 1, 40, 100, 20)
26         self.set_exp(15)
27
28     def special_ability(self, target):
29         print(f"→ {self.name} ударил клювом! {target.name}!")
30         target.apply_effect("Кровотечение", duration=2, damage_per_turn=3)
31
32
33 class AlwaysComeBack(Enemy):
34     def __init__(self):
35         super().__init__("Человек в маске", 200, 20, 10, 10, 200, 120)
36         self.set_exp(100)
37         self.resurrection_used = False
38         self.resurrect_chance = 0.6
39
40     def take_damage(self, damage):
41         actual = super().take_damage(damage)
42         if self.health <= 0 and not self.resurrection_used:
43             if random.random() < self.resurrect_chance:
44                 self.resurrection_used = True
45                 self.health = self.max_health // 2
46                 print(f"☐ {self.name} пал... но ВОСКРЕС!")
47                 print(f"☐ {self.name} теперь имеет {self.health} HP!")
48             else:
49                 print(f"☐ {self.name} окончательно уничтожен.")
50         return actual

```

Рисунок 4 – Листинг `enemy1.py`

Зачем это нужно: Уникальные способности врагов (кровотечение, воскрешение) ломают шаблонный бой и заставляют игрока адаптировать тактику. Это предотвращает монотонность и повышает вовлечённость. Базовый класс `Enemy` обеспечивает единообразие при добавлении новых типов противников.

2.5 Враги – продвинутые типы и генерация

Реализованы три уникальных врага и функция генерации с учётом этажа.

На рисунке 5 – код.

```

1 import random
2
3 # === Враги – часть 2 ===
4
5 class Regenerator(Enemy):
6     def __init__(self):
7         super().__init__("Регенератор", 60, 15, 4, 45, 150, 40)
8         self.set_exp(30)
9
10    def special_ability(self, target):
11        self.heal(8)
12
13
14 class Vsosun(Enemy):
15     def __init__(self):
16         super().__init__("Всасыватель", 100, 20, 2, 10, 170, 65)
17         self.set_exp(50)
18
19     def special_ability(self, target):
20         heal = min(10, target.damage // 2)
21         self.health += heal
22         print(f" → {self.name} высасывает {heal} HP из {target.name}!")
23
24
25 class PowerUpper(Enemy):
26     def __init__(self):
27         super().__init__("Неизвестный", 50, 5, 10, 50, 170, 60)
28         self.set_exp(70)
29
30     def special_ability(self, target):
31         old_damage = self.damage
32         self.damage = int(self.damage * 1.5)
33         print(f" → {self.name} вы видите, как у него неожиданно появляются мышцы...")
34         print(f" → Урон: {old_damage} → {self.damage}")
35
36
37 def generate_enemy(floor):
38     enemies = [Vsosun(), Regenerator(), PowerUpper(), AlwaysComeBack(), PingVin735()]
39     enemy = random.choice(enemies)
40     enemy.health += floor * 5
41     enemy.damage += floor * 2
42     enemy.exp_reward += floor * 3
43     return enemy

```

Рисунок 5 – Листинг enemy2.py

Зачем это нужно: Разнообразие врагов (вампиризм, усиление) создаёт непредсказуемость и риск. Функция `generate_enemy` с динамическим усилением по этажам гарантирует, что игра остаётся вызовом на протяжении всего прохождения, соответствующим принципу прогрессивной сложности.

2.6 Расы – ядро уникальных механик

Реализованы три уникальные расы с особыми механиками: Поглотитель (впитывает урон), Гуль (лечится при атаке), Пробуждённый (усиливается в бою).

На рисунке 6 – код.

```

1 import random
2
3 # === PACbl ===
4 class Absorber(Character):
5     def __init__(self, name):
6         hp = random.randint(110, 130)
7         attack = random.randint(14, 16)
8         defense = random.randint(7, 9)
9         agility = random.randint(28, 32)
10        height = random.randint(180, 200)
11        weight = random.randint(80, 90)
12        super().__init__(name, "Поглотитель", hp, attack, defense, agility, height, weight)
13        self.absorbed_damage = 0
14
15    def take_damage(self, damage):
16        if random.random() < self.evasion_chance:
17            print(f" → {self.name} уклонился!")
18            return 0
19
20        actual_damage = max(1, damage - self.total_defense)
21        absorbed = actual_damage // 2
22        real_taken = actual_damage - absorbed
23
24        self.absorbed_damage += absorbed
25        self.health -= real_taken
26        if self.health < 0:
27            self.health = 0
28
29        print(f" → {self.name} поглотил {absorbed} урона! Получил только {real_taken}.")
30        print(f" → Накоплено для выплеска: {self.absorbed_damage}")
31        return actual_damage
32
33    def attack(self, other):
34        base_dmg = self.total_attack
35        bonus_dmg = self.absorbed_damage
36
37        if bonus_dmg > 0:
38            total_dmg = base_dmg + bonus_dmg
39            print(f"{self.name} бьёт {other.name} с силой {base_dmg} + {bonus_dmg} накопленного(!)")
40            other.take_damage(total_dmg)
41            self.absorbed_damage = 0
42        else:
43            print(f"{self.name} бьёт {other.name}!")
44            other.take_damage(base_dmg)
45
46
47 class Ghoul(Character):
48     def __init__(self, name):
49         hp = random.randint(90, 110)
50         attack = random.randint(18, 22)
51         defense = random.randint(4, 6)
52         agility = random.randint(48, 52)
53         height = random.randint(170, 180)
54         weight = random.randint(55, 65)
55         super().__init__(name, "Гуль", hp, attack, defense, agility, height, weight)
56
57    def attack(self, other):
58        print(f"{self.name} бьёт {other.name}!")
59        other.take_damage(self.total_attack)
60        heal_amount = min(8, self.total_attack // 3)
61        self.heal(heal_amount)
62
63
64 class Awake(Character):
65     def __init__(self, name):
66         hp = random.randint(85, 95)
67         attack = random.randint(23, 27)
68         defense = random.randint(2, 4)
69         agility = random.randint(68, 72)
70         height = random.randint(175, 185)
71         weight = random.randint(65, 75)
72         super().__init__(name, "Пробуждённый", hp, attack, defense, agility, height, weight)
73         self.battle_actions = 0
74
75    def attack(self, other):
76        print(f"{self.name} бьёт {other.name}!")
77        other.take_damage(self.total_attack)
78        self.damage += 2
79        self.agility += 1
80        self.battle_actions += 1
81        print(f" → {self.name} пробуждается! Атака +2, Ловкость +1")
82
83    def take_damage(self, damage):
84        actual = super().take_damage(damage)
85        if actual > 0:
86            self.defense += 1
87            self.battle_actions += 1
88            print(f" → Боль делает {self.name} сильнее! Защита +1")
89        return actual

```

Рисунок 6 – Листинг races_core.py

Зачем это нужно: Уникальные расы с разными механиками дают игроку осмысленный выбор при старте. Случайная генерация характеристик в рамках расы обеспечивает реиграбельность: каждый запуск — новый опыт.

2.7 Создание персонажа

Реализована логика выбора расы и генерации случайных характеристик в допустимых пределах.

На рисунке 7 — код.

```

1 # Этот файл подключается в main_game.py
2 # Логика выбора расы и генерации характеристик уже встроена в main_game.py
3 # Но если выделить отдельно:
4
5 def create_character(choice, name):
6     if choice == "1":
7         player = Absorber(name)
8     elif choice == "2":
9         player = Ghoul(name)
10    else:
11        player = Awake(name)
12    return player

```

Рисунок 7 — Листинг `character_creation.py`

Зачем это нужно: Процесс создания персонажа задаёт начальные условия игры. Случайность в рамках расы обеспечивает баланс между предсказуемостью и вариативностью, что соответствует духу RPG.

2.8 Основной игровой цикл

Реализован главный цикл: исследование подземелья, выбор пути, обработка типов комнат (бой, сундук, отдых).

На рисунке 8 — код.

```

1 import random
2
3 # =====
4 # ОСНОВНАЯ ИГРА
5 # =====
6
7 def main():
8     print("☞ ДОБРО ПОЖАЛОВАТЬ В ПОДЗЕМЕЛЬЕ БОЛИ ☞")
9     print("Только сильнейшие выживут. Остальные – корм для пингвинов.\n")
10
11     print("Выбери свою суть:")
12     print("1. Поглотитель впитывает( урон и выплёскивает обратно)")
13     print("2. Гуль лечится( при каждой атаке)")
14     print("3. Пробуждённый становится( сильнее в бою)")
15
16     while True:
17         choice = input("Твой выбор (1-3): ").strip()
18         if choice in ("1", "2", "3"):
19             break
20         print("Цифру!")
21
22     name = input("Имя твоего аватара боли: ").strip() or "Безымянный"
23
24     if choice == "1":
25         player = Absorber(name)
26     elif choice == "2":
27         player = Ghoul(name)
28     else:
29         player = Awake(name)
30
31     print(f"\n {player.name}, {player.race}")
32     print("Перед тобой бесконечные коридоры... Выбирай путь!\n")
33
34     floor = 1
35     room = 0
36     inventory = Inventory()
37
38     while player.is_alive():
39         room += 1
40         print(f"\n{' '*50}")
41         print(f"ЭТАЖ {floor} • КОМНАТА {room}")
42         print(f"HP: {player.health}/{player.max_health} | Урон: {player.total_attack}")
43
44         left_room = random.choice(["enemy", "chest", "rest"])
45         right_room = random.choice(["enemy", "chest", "rest"])
46
47         left_known = random.choice([True, False])
48         right_known = random.choice([True, False])
49
50         print("\nПеред тобой развилка:")
51         left_desc = left_room if left_known else "???"
52         right_desc = right_room if right_known else "???"
53         print(f"(1) СЛЕВА: {left_desc}")
54         print(f"(2) СПРАВА: {right_desc}")
55
56         while True:
57             path = input("\nКуда двинешь? (1/2): ").strip()
58             if path in ("1", "2"):
59                 break
60             print("1 или 2, епта!")
61
62         chosen_room = left_room if path == "1" else right_room
63
64         if chosen_room == "enemy":
65             enemy = generate_enemy(floor)
66             print(f"☞ Вызов! {enemy.name} бросает тебе вызов!")
67
68             while player.is_alive() and enemy.is_alive():
69                 print("\n--- ТВОЙ ХОД ---")
70                 print("1. Атаковать")
71                 print("2. Использовать предмет")
72                 print("3. Сбежать (50%)")
73
74                 action = input("Действие: ").strip()
75
76                 if action == "1":
77                     player.attack(enemy)
78                     enemy.update_effects()
79                 elif action == "2":
80                     inventory.show()
81                     if inventory.items:
82                         try:
83                             idx = int(input("Номер предмета (0 – отмена): "))
84                             if idx > 0:
85                                 inventory.use_item(idx, player)
86                                 continue
87                             except ValueError:
88                                 print("Число давай!")
89                                 continue
90                     else:
91                         print("Инвентарь пуст!")
92                         continue
93                 elif action == "3":
94                     if random.random() < 0.5:
95                         print("Ты сбежал, как крыса!")
96                         break
97                     else:
98                         print("Не вышло... Они тебя настигли!")
99                 else:
100                     print("Не вышло... Они тебя настигли!")

```

Зачем это нужно: Основной цикл объединяет все игровые системы в единый процесс. Развилки с частичной видимостью создают напряжение и стратегическое планирование, что является основой исследования в RPG.

2.9 Боевая система и механики

Реализованы пошаговый бой, действия игрока (атака, предмет, побег), обработка эффектов, генерация лута.

На рисунке 9 – код.

```

1 import random
2
3 def generate_loot():
4     loot_pool = [
5         {"name": "Зелье лечения", "type": "potion", "heal": 25},
6         {"name": "Меч новичка", "type": "weapon", "attack": 5},
7         {"name": "Стальной меч", "type": "weapon", "attack": 8},
8         {"name": "Кожаная броня", "type": "armor", "defense": 3},
9         {"name": "Кольчуга", "type": "armor", "defense": 5},
10        {"name": "Монеты", "type": "gold", "amount": random.randint(10, 30)}
11    ]
12    return random.choice(loot_pool)
13
14 def conduct_battle(player, enemy, inventory, floor):
15     while player.is_alive() and enemy.is_alive():
16         print("\n--- ТВОЙ ХОД ---")
17         print("1. Атаковать")
18         print("2. Использовать предмет")
19         print("3. Сбежать (50%)")
20
21         action = input("Действие: ").strip()
22
23         if action == "1":
24             player.attack(enemy)
25             enemy.update_effects()
26         elif action == "2":
27             inventory.show()
28             if inventory.items:
29                 try:
30                     idx = int(input("Номер предмета (0 – отмена): "))
31                     if idx > 0:
32                         inventory.use_item(idx, player)
33                     continue
34                 except ValueError:
35                     print("Число давай!")
36                     continue
37             else:
38                 print("Инвентарь пуст!")
39                 continue
40         elif action == "3":
41             if random.random() < 0.5:
42                 print("Ты сбежал, как крыса!")
43                 return
44             else:
45                 print("Не вышло... Они тебя настигли!")
46         else:
47             print("Пропуск хода!")
48
49         if enemy.is_alive():
50             print(f"\n--- ХОД ВРАГА ---")
51             enemy.attack(player)
52             player.update_effects()
53
54     if player.is_alive():
55         print(f"\n Победа! Получено {enemy.exp_reward} опыта.")
56         player.gain_exp(enemy.exp_reward)
57         inventory.gold += random.randint(5, 15)
58         if random.random() < 0.4:
59             loot = generate_loot()
60             if loot["type"] == "gold":
61                 inventory.gold += loot["amount"]
62             print(f"Найдено {loot['amount']} золота!")
63         else:
64             inventory.add_item(loot)

```

Рисунок 9 – Листинг game_mechanics.py

Зачем это нужно: Боевая система – сердце RPG. Пошаговые действия, уклонение, эффекты и добыча создают глубокую тактическую игру, где каждое решение имеет значение.

3 Заключение

В ходе выполнения лабораторной работы №8 был успешно реализован прототип консольной текстовой RPG-игры, полностью соответствующий поставленному заданию.

Программа демонстрирует все ключевые игровые механики: создание персонажа одной из трёх уникальных рас с генерацией случайных характеристик в рамках выбранной расы, исследование подземелья, состоящего из случайных комнат с частичной видимостью событий, пошаговую боевую систему с учётом уклонения и особых способностей, а также систему прокачки и управления инвентарём.

Были реализованы три расы персонажей – «Поглотитель», «Гуль» и «Пробуждённый» – каждая из которых обладает уникальной игровой механикой, влияющей на тактику ведения боя.

Враги также наделены особыми способностями, такими как воскрешение, вампиризм и усиление в бою, что добавляет разнообразие и сложность игровому процессу.

Сложность игры динамически возрастает с каждым новым этажом подземелья.

Разработанное приложение представляет собой законченный, работоспособный продукт, написанный на языке Python с использованием принципов объектно-ориентированного программирования.

Код структурирован, читаем и легко поддаётся расширению.

Все поставленные задачи выполнены в полном объёме, что подтверждает успешное освоение материала по дисциплине «Информатика и программирование».