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

Владивосток 2025

Содержание

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.

1.1 Принципы проектирования

При разработке RPG–игры были применены следующие принципы объектно–ориентированного проектирования:

Единственная ответственность (SRP): Каждый класс решает одну задачу. Character отвечает за характеристики и взаимодействие, Inventory – за предметы, Enemy – за поведение врагов.

Открытость/Закрытость (OCP): Система легко расширяема. Для добавления нового врага достаточно создать новый класс, наследуясь от Enemy, без изменения существующего кода.

Наследование: Все игровые сущности (игрок, враги) наследуются от базового класса Character, что обеспечивает единообразие интерфейса и повторное использование кода.

Инкапсуляция: Внутреннее состояние объектов (здоровье, урон, эффекты) скрыто, а доступ осуществляется через методы, что предотвращает некорректное изменение данных.

Модульность: Код разбит на логические модули (character, inventory, enemies, game mechanics), что упрощает поддержку, тестирование и понимание структуры проекта.

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

```python
import random

class Character:
    def __init__(self, name, race, health, damage, defense, agility, height, weight):
        self.name = name
        self.race = race
        self.health = health
        self.max_health = health
        self.damage = damage
        self.defense = defense
        self.agility = agility
        self.height = height
        self.weight = weight

        self.level = 1
        self.exp = 0
        self.stat_points = 0

        self.weapon_name = "Палка"
        self.weapon_attack = 0
        self.armor_name = "Бандаж"
        self.armor_defense = 0

        self.effects = []

    @property
    def index_mass(self):
        height_m = self.height / 100
        return self.weight / (height_m ** 2)

    @property
    def total_attack(self):
        return self.damage + self.weapon_attack

    @property
    def total_defense(self):
        imt = self.index_mass
        bonus_defense = 0
        if imt < 18.5:
            bonus_defense = -3
        elif imt > 25:
            bonus_defense = 5
        return self.defense + self.armor_defense + bonus_defense

    @property
    def evasion_chance(self):
        base_dodge = self.agility / 100
        imt = self.index_mass
        bonus_dodge = 0
        if imt < 18.5:
            bonus_dodge = 0.2
        elif 18.5 <= imt <= 25:
            bonus_dodge = 0.05
        else:
            bonus_dodge = -0.15
        total = base_dodge + bonus_dodge
        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

```
    def apply_effect(self, effect_name, duration, damage_per_turn=0):
        self.effects.append({
            "name": effect_name,
            "duration": duration,
            "damage_per_turn": damage_per_turn
        })

    def update_effects(self):
        new_effects = []
        for effect in self.effects:
            if effect["duration"] > 0:
                if effect["damage_per_turn"] > 0:
                    print(f"  → {self.name} получает {effect['damage_per_turn']} урона от '{effect['name']}'")
                    self.take_damage(effect["damage_per_turn"])
                effect["duration"] -= 1
                new_effects.append(effect)
            else:
                print(f"  → Эффект '{effect['name']}' закончился.")
        self.effects = new_effects

    def take_damage(self, damage):
        if random.random() < self.evasion_chance:
            print(f"  → {self.name} уклонился!")
            return 0

        actual_damage = max(1, damage - self.total_defense)
        self.health -= actual_damage
        if self.health < 0:
            self.health = 0
        print(f"  → {self.name} получил {actual_damage} урона. Здоровье: {self.health}")
        return actual_damage

    def attack(self, other):
        print(f"{self.name} бьёт {other.name}!")
        other.take_damage(self.total_attack)

    def is_alive(self):
        return self.health > 0

    def heal(self, amount):
        old_health = self.health
        self.health = min(self.max_health, self.health + amount)
        healed = self.health - old_health
        print(f"  → {self.name} восстановил {healed} HP. Теперь HP: {self.health}")

    def gain_exp(self, amount):
        self.exp += amount
        needed = self.level * 50
        if self.exp >= needed:
            self.level_up()

    def level_up(self):
        self.level += 1
        self.stat_points += 3
        print(f"\n Уровень повышен! Теперь у вас {self.level}- уровень. Получено 3 очка характеристик.")
        self.apply_statpoints()

    def apply_statpoints(self):
        while self.stat_points > 0:
            print(f"\Уп вас {self.stat_points} очкоаов(/). Куда вложите?")
            print("1. +5 Здоровья")
            print("2. +1 Защиты")
            print("3. +1 Атаки")
            print("4. +1 Ловкости")

            choice = input("Выбор (1-4): ").strip()
            if choice == "1":
                self.max_health += 5
                self.health += 5
                self.stat_points -= 1
            elif choice == "2":
                self.defense += 1
                self.stat_points -= 1
            elif choice == "3":
                self.damage += 1
                self.stat_points -= 1
            elif choice == "4":
                self.agility += 1
                self.stat_points -= 1
            else:
                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

```
class Inventory:
    def __init__(self, capacity=10):
        self.items = []
        self.capacity = capacity
        self.gold = 0

    def add_item(self, item):
        if len(self.items) >= self.capacity:
            print("Сори инвентарь полон, попробуй чтото- выкинуть:")
            self.show()
            idx = input("Номер предмета для выброса ничего(0-): ")
            if idx.isdigit() and 1 <= int(idx) <= len(self.items):
                self.items.pop(int(idx)-1)
            else:
                print("Предмет не добавлен")
                return False
        self.items.append(item)
        print(f"Получен предмет: {item.get('name', '???')}")
        return True

    def show(self):
        print("\nИнвентарь--------")
        print(f"Золото: {self.gold}")
        if not self.items:
            print("Пусто.")
        else:
            for i, item in enumerate(self.items, 1):
                print(f"{i}. {item['name']}")

    def use_item(self, index, character):
        if 1 <= index <= len(self.items):
            item = self.items[index - 1]
            if item["type"] == "potion":
                character.heal(item["heal"])
                self.items.pop(index - 1)
            elif item["type"] == "weapon":
                old_name, old_atk = character.weapon_name, character.weapon_attack
                character.weapon_name = item["name"]
                character.weapon_attack = item["attack"]
                self.items.pop(index - 1)
                if old_name != "Палка":
                    self.items.append({"name": old_name, "type": "weapon", "attack": old_atk})
                print(f"Экипировано: {item['name']}")
            elif item["type"] == "armor":
                old_name, old_def = character.armor_name, character.armor_defense
                character.armor_name = item["name"]
                character.armor_defense = item["defense"]
                self.items.pop(index - 1)
                if old_name != "Бандаж":
                    self.items.append({"name": old_name, "type": "armor", "defense": old_def})
                print(f"Экипировано: {item['name']}")
            else:
                print("Нельзя использовать.")
        else:
            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 мощью.

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

```python
import random

# === Враги — часть 1 ===

class Enemy(Character):
    def __init__(self, name, hp, attack, defense, agility, height=170, weight=60):
        super().__init__(name, "Монстр", hp, attack, defense, agility, height, weight)
        self.exp_reward = 0

    def set_exp(self, exp):
        self.exp_reward = exp

    def special_ability(self, target):
        pass

    def attack(self, other):
        print(f"{self.name} бьёт {other.name}!")
        other.take_damage(self.total_attack)
        if random.random() < 0.3:
            self.special_ability(other)


class PingVin735(Enemy):
    def __init__(self):
        super().__init__("Пингвин", 25, 6, 1, 40, 100, 20)
        self.set_exp(15)

    def special_ability(self, target):
        print(f"  → {self.name} ударил клювом! {target.name}!")
        target.apply_effect("Кровотечение", duration=2, damage_per_turn=3)


class AlwaysComeBack(Enemy):
    def __init__(self):
        super().__init__("Человек в маске", 200, 20, 10, 10, 200, 120)
        self.set_exp(100)
        self.resurrection_used = False
        self.resurrect_chance = 0.6

    def take_damage(self, damage):
        actual = super().take_damage(damage)
        if self.health <= 0 and not self.resurrection_used:
            if random.random() < self.resurrect_chance:
                self.resurrection_used = True
                self.health = self.max_health // 2
                print(f"✨ {self.name} пал... но ВОСКРЕС!")
                print(f"✨ {self.name} теперь имеет {self.health} HP!")
            else:
                print(f"✨ {self.name} окончательно уничтожен.")
        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

```python
import random

# === Враги – часть 2 ===

class Regenerator(Enemy):
    def __init__(self):
        super().__init__("Регенератор", 60, 15, 4, 45, 150, 40)
        self.set_exp(30)

    def special_ability(self, target):
        self.heal(8)


class Vsosun(Enemy):
    def __init__(self):
        super().__init__("Всасыватель", 100, 20, 2, 10, 170, 65)
        self.set_exp(50)

    def special_ability(self, target):
        heal = min(10, target.damage // 2)
        self.health += heal
        print(f"  → {self.name} высасывает {heal} HP из {target.name}!")


class PowerUpper(Enemy):
    def __init__(self):
        super().__init__("Неизвестный", 50, 5, 10, 50, 170, 60)
        self.set_exp(70)

    def special_ability(self, target):
        old_damage = self.damage
        self.damage = int(self.damage * 1.5)
        print(f"  → {self.name} вы видите, как у него неожиданно появляются мышцы...")
        print(f"  → Урон: {old_damage} → {self.damage}")


def generate_enemy(floor):
    enemies = [Vsosun(), Regenerator(), PowerUpper(), AlwaysComeBack(), PingVin735()]
    enemy = random.choice(enemies)
    enemy.health += floor * 5
    enemy.damage += floor * 2
    enemy.exp_reward += floor * 3
    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

```python
import random

# === РАСЫ ===
class Absorber(Character):
    def __init__(self, name):
        hp = random.randint(110, 130)
        attack = random.randint(14, 16)
        defense = random.randint(7, 9)
        agility = random.randint(28, 32)
        height = random.randint(180, 200)
        weight = random.randint(80, 90)
        super().__init__(name, "Поглотитель", hp, attack, defense, agility, height, weight)
        self.absorbed_damage = 0

    def take_damage(self, damage):
        if random.random() < self.evasion_chance:
            print(f"  → {self.name} уклонился!")
            return 0

        actual_damage = max(1, damage - self.total_defense)
        absorbed = actual_damage // 2
        real_taken = actual_damage - absorbed

        self.absorbed_damage += absorbed
        self.health -= real_taken
        if self.health < 0:
            self.health = 0

        print(f"  → {self.name} поглотил {absorbed} урона! Получил только {real_taken}.")
        print(f"  → Накоплено для выплеска: {self.absorbed_damage}")
        return actual_damage

    def attack(self, other):
        base_dmg = self.total_attack
        bonus_dmg = self.absorbed_damage

        if bonus_dmg > 0:
            total_dmg = base_dmg + bonus_dmg
            print(f"{self.name} бьёт {other.name} с силой {base_dmg} + {bonus_dmg} накопленного()!")
            other.take_damage(total_dmg)
            self.absorbed_damage = 0
        else:
            print(f"{self.name} бьёт {other.name}!")
            other.take_damage(base_dmg)


class Ghoul(Character):
    def __init__(self, name):
        hp = random.randint(90, 110)
        attack = random.randint(18, 22)
        defense = random.randint(4, 6)
        agility = random.randint(48, 52)
        height = random.randint(170, 180)
        weight = random.randint(55, 65)
        super().__init__(name, "Гуль", hp, attack, defense, agility, height, weight)

    def attack(self, other):
        print(f"{self.name} бьёт {other.name}!")
        other.take_damage(self.total_attack)
        heal_amount = min(8, self.total_attack // 3)
        self.heal(heal_amount)


class Awake(Character):
    def __init__(self, name):
        hp = random.randint(85, 95)
        attack = random.randint(23, 27)
        defense = random.randint(2, 4)
        agility = random.randint(68, 72)
        height = random.randint(175, 185)
        weight = random.randint(65, 75)
        super().__init__(name, "Пробуждённый", hp, attack, defense, agility, height, weight)
        self.battle_actions = 0

    def attack(self, other):
        print(f"{self.name} бьёт {other.name}!")
        other.take_damage(self.total_attack)
        self.damage += 2
        self.agility += 1
        self.battle_actions += 1
        print(f"  → {self.name} пробуждается! Атака +2, Ловкость +1")

    def take_damage(self, damage):
        actual = super().take_damage(damage)
        if actual > 0:
            self.defense += 1
            self.battle_actions += 1
            print(f"  → Боль делает {self.name} сильнее! Защита +1")
        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

```
# Этот файл подключается в main_game.py
# Логика выбора расы и генерации характеристик уже встроена в main_game.py
# Но если выделить отдельно:

def create_character(choice, name):
    if choice == "1":
        player = Absorber(name)
    elif choice == "2":
        player = Ghoul(name)
    else:
        player = Awake(name)
    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

```python
import random

# ======================
# ОСНОВНАЯ ИГРА
# ======================

def main():
    print("□ ДОБРО ПОЖАЛОВАТЬ В ПОДЗЕМЕЛЬЕ БОЛИ □")
    print("Только сильнейшие выживут. Остальные — корм для пингвинов.\n")

    print("Выбери свою суть:")
    print("1. Поглотитель впитывает( урон и выплёскивает обратно)")
    print("2. Гуль лечится( при каждой атаке)")
    print("3. Пробуждённый становится( сильнее в бою)")

    while True:
        choice = input("Твой выбор (1-3): ").strip()
        if choice in ("1", "2", "3"):
            break
        print("Цифру!")

    name = input("Имя твоего аватара боли: ").strip() or "Безымянный"

    if choice == "1":
        player = Absorber(name)
    elif choice == "2":
        player = Ghoul(name)
    else:
        player = Awake(name)

    print(f"\n {player.name}, {player.race}")
    print("Перед тобой бесконечные коридоры... Выбирай путь!\n")

    floor = 1
    room = 0
    inventory = Inventory()

    while player.is_alive():
        room += 1
        print(f"\n{'='*50}")
        print(f"ЭТАЖ {floor} • КОМНАТА {room}")
        print(f"HP: {player.health}/{player.max_health} | Урон: {player.total_attack}")

        left_room = random.choice(["enemy", "chest", "rest"])
        right_room = random.choice(["enemy", "chest", "rest"])

        left_known = random.choice([True, False])
        right_known = random.choice([True, False])

        print("\Передn тобой развилка:")
        left_desc = left_room if left_known else "???"
        right_desc = right_room if right_known else "???"
        print(f"(1) СЛЕВА: {left_desc}")
        print(f"(2) СПРАВА: {right_desc}")

        while True:
            path = input("\Кудап двинешь? (1/2): ").strip()
            if path in ("1", "2"):
                break
            print("1 или 2, епта!")

        chosen_room = left_room if path == "1" else right_room

        if chosen_room == "enemy":
            enemy = generate_enemy(floor)
            print(f"\□n  ВЫЗОВ! {enemy.name} бросает тебе вызов!")

            while player.is_alive() and enemy.is_alive():
                print("\n--- ТВОЙ ХОД ---")
                print("1. Атаковать")
                print("2. Использовать предмет")
                print("3. Сбежать (50%)")

                action = input("Действие: ").strip()

                if action == "1":
                    player.attack(enemy)
                    enemy.update_effects()
                elif action == "2":
                    inventory.show()
                    if inventory.items:
                        try:
                            idx = int(input("Номер предмета (0 — отмена): "))
                            if idx > 0:
                                inventory.use_item(idx, player)
                            continue
                        except ValueError:
                            print("Число давай!")
                            continue
                    else:
                        print("Инвентарь пуст!")
                        continue
                elif action == "3":
                    if random.random() < 0.5:
                        print("Ты сбежал, как крыса!")
                        break
                    else:
                        print("Не вышло... Они тебя настигли!")
                else:
```

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

```python
import random

def generate_loot():
    loot_pool = [
        {"name": "Зелье лечения", "type": "potion", "heal": 25},
        {"name": "Меч новичка", "type": "weapon", "attack": 5},
        {"name": "Стальной меч", "type": "weapon", "attack": 8},
        {"name": "Кожаная броня", "type": "armor", "defense": 3},
        {"name": "Кольчуга", "type": "armor", "defense": 5},
        {"name": "Монеты", "type": "gold", "amount": random.randint(10, 30)}
    ]
    return random.choice(loot_pool)

def conduct_battle(player, enemy, inventory, floor):
    while player.is_alive() and enemy.is_alive():
        print("\n--- ТВОЙ ХОД ---")
        print("1. Атаковать")
        print("2. Использовать предмет")
        print("3. Сбежать (50%)")

        action = input("Действие: ").strip()

        if action == "1":
            player.attack(enemy)
            enemy.update_effects()
        elif action == "2":
            inventory.show()
            if inventory.items:
                try:
                    idx = int(input("Номер предмета (0 – отмена): "))
                    if idx > 0:
                        inventory.use_item(idx, player)
                    continue
                except ValueError:
                    print("Число давай!")
                    continue
            else:
                print("Инвентарь пуст!")
                continue
        elif action == "3":
            if random.random() < 0.5:
                print("Ты сбежал, как крыса!")
                return
            else:
                print("Не вышло... Они тебя настигли!")
        else:
            print("Пропуск хода!")

        if enemy.is_alive():
            print(f"\n--- ХОД ВРАГА ---")
            enemy.attack(player)
            player.update_effects()

    if player.is_alive():
        print(f"\n Победа! Получено {enemy.exp_reward} опыта.")
        player.gain_exp(enemy.exp_reward)
        inventory.gold += random.randint(5, 15)
        if random.random() < 0.4:
            loot = generate_loot()
            if loot["type"] == "gold":
                inventory.gold += loot["amount"]
                print(f"Найдено {loot['amount']} золота!")
            else:
                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.