Ниже приведён перечень учебных программ в игровой тематике, ориентированных на школьников разного возраста и уровня подготовки. Все варианты спроектированы так, чтобы естественным образом отрабатывать базовые принципы ООП в C++: инкапсуляцию, наследование, полиморфизм и композицию. Формат каждой задачи — название и развёрнутое описание с акцентом на структуру программы и обучающую ценность.

1. Академия героев

Суть задачи Небольшая консольная ролевая игра, в которой игрок управляет персонажами-учениками магической академии.

Описание реализации Создаётся базовый класс Character, содержащий общие характеристики: имя, уровень, здоровье, энергия. От него наследуются классы Mage, Warrior, Archer и т.п., каждый из которых переопределяет методы атаки или способностей.

ООП-аспекты

- Наследование для специализаций персонажей
- Полиморфизм при использовании атак через указатель на базовый класс
- Инкапсуляция характеристик через методы доступа

Учебный результат Студенты видят, как в играх создаются разные типы персонажей на основе общего шаблона.

2. Монстры и охотники

Суть задачи Симуляция охоты на монстров, где разные охотники используют разные типы оружия.

Описание реализации Реализуется иерархия классов Monster (например, Dragon, Goblin, Slime) и классов оружия Weapon (Sword, Bow, MagicStaff). Охотник содержит оружие как объект (композиция).

ООП-аспекты

- Композиция (охотник “имеет” оружие)
- Абстрактный класс или интерфейс Weapon
- Полиморфное поведение оружия при атаке

Учебный результат Понимание различий между наследованием и композицией на игровом примере.

3. Фэнтезийный зоопарк

Суть задачи Управление виртуальным зоопарком магических существ.

Описание реализации Создаётся базовый класс Creature с виртуальным методом makeSound() и eat(). Производные классы: Dragon, Unicorn, Phoenix. Все существа хранятся в одном контейнере.

ООП-аспекты

- Виртуальные функции
- Полиморфизм при работе с коллекцией существ
- Переопределение поведения

Учебный результат Студенты учатся работать с базовыми классами и понимать смысл виртуальных методов.

4. Инвентарь героя

Суть задачи Система инвентаря для персонажа в игре.

Описание реализации Реализуется базовый класс Item с производными: Potion, Weapon, Armor. Герой может подбирать, хранить и использовать предметы.

ООП-аспекты

- Базовый класс для предметов
- Переопределение метода use()
- Работа с контейнерами объектов

Учебный результат Формируется понимание того, как игровые предметы описываются в коде.

5. Арена пошаговых боёв

Суть задачи Простейшая пошаговая боевая система.

Описание реализации Игрок выбирает бойца, затем бой проходит по ходам. Каждый боец — объект класса Fighter, от которого наследуются разные типы бойцов с уникальными умениями.

ООП-аспекты

- Наследование и переопределение логики хода
- Полиморфизм при обработке действий бойцов
- Разделение логики боя и логики персонажей

Учебный результат Понимание архитектуры игры и распределения ответственности между классами.

6. Школа магии

Суть задачи Симуляция обучения магов различным заклинаниям.

Описание реализации Класс Spell с виртуальным методом cast(). Маги изучают заклинания и хранят их в своём списке. Заклинания различаются по эффекту.

ООП-аспекты

- Полиморфные объекты заклинаний
- Динамическое добавление новых умений
- Инкапсуляция логики заклинаний

Учебный результат Демонстрация расширяемости системы без изменения существующего кода.

7. Гильдия приключенцев

Суть задачи Управление командой персонажей, выполняющих задания.

Описание реализации Есть класс Quest и класс Adventurer. Разные задания требуют разных характеристик. Команда формируется из объектов разных классов.

ООП-аспекты

- Использование абстракций
- Взаимодействие объектов разных типов
- Простейшая логика принятия решений

Учебный результат Понимание объектного взаимодействия в игровых системах.

8. Башня подземелий

Суть задачи Прохождение башни с этажами, на каждом из которых разное событие.

Описание реализации Класс Room с наследниками: EnemyRoom, TreasureRoom, TrapRoom. Башня — набор комнат, обрабатываемых последовательно.

ООП-аспекты

- Полиморфизм при обработке комнат
- Абстрактный базовый класс
- Чёткое разделение логики

Учебный результат Формирование навыков проектирования иерархий классов.

9. Карточная фэнтези-игра

Суть задачи Простейшая карточная игра с существами и заклинаниями.

Описание реализации Карты представлены базовым классом Card, от которого наследуются CreatureCard и SpellCard. Каждая карта имеет собственное поведение.

ООП-аспекты

- Полиморфная обработка карт
- Работа с динамическими объектами
- Инкапсуляция игровой логики

Учебный результат Понимание того, как реализуются карточные игры на уровне кода.

10. Фэнтези-симулятор NPC

Суть задачи Симуляция поведения неигровых персонажей в деревне.

Описание реализации Класс NPC и производные: Trader, Guard, Farmer. Каждый NPC по-разному реагирует на игрока.

ООП-аспекты

- Наследование поведения
- Переопределение методов взаимодействия
- Моделирование игрового мира

Учебный результат Понимание применения ООП для моделирования игровых сущностей.

Пример реализации - Фэнтезийный зоопарк

Цель учебной задачи

Показать:

- базовый класс;
- наследование;
- виртуальные методы;
- полиморфизм при работе с коллекцией объектов.

План реализации (пошагово)

Шаг 1. Анализ предметной области

В зоопарке есть существа. Все существа:

- имеют имя;
- могут издавать звук;
- могут есть.

Но каждое существо делает это по-своему.

Шаг 2. Проектирование классов

Базовый класс:

- Creature
  - поле: name
  - методы:
    - makeSound() — виртуальный
    - eat() — виртуальный

Наследники:

- Dragon
- Unicorn
- Phoenix

Шаг 3. Определение взаимодействия

- Все существа хранятся в одном массиве (через указатели на Creature)
- Вызываем методы, не зная точного типа существа

Шаг 4. Реализация и тестирование

- Создаём несколько существ
- Добавляем в контейнер
- Вызываем методы в цикле

Полная реализация кода (C++)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Creature {
protected:
    string name;

public:
    Creature(string name) : name(name) {}

    virtual void makeSound() {
        cout << name << " издает странный звук." << endl;
    }

    virtual void eat() {
        cout << name << " ест еду." << endl;
    }

    virtual ~Creature() {}
};

class Dragon : public Creature {
public:
    Dragon(string name) : Creature(name) {}

    void makeSound() override {
        cout << name << " рычит и изрыгает пламя!" << endl;
    }

    void eat() override {
        cout << name << " ест мясо." << endl;
    }
};
```

```
class Unicorn : public Creature {
public:
    Unicorn(string name) : Creature(name) {}

    void makeSound() override {
        cout << name << " тихо фыркает." << endl;
    }

    void eat() override {
        cout << name << " ест магические травы." << endl;
    }
};

class Phoenix : public Creature {
public:
    Phoenix(string name) : Creature(name) {}

    void makeSound() override {
        cout << name << " издает огненный крик." << endl;
    }

    void eat() override {
        cout << name << " питается пламенем." << endl;
    }
};

int main() {
    vector<Creature*> zoo;

    zoo.push_back(new Dragon("Дракон Огнекрыл"));
    zoo.push_back(new Unicorn("Единорог Луна"));
    zoo.push_back(new Phoenix("Феникс Искра"));

    for (Creature* creature : zoo) {
        creature->makeSound();
        creature->eat();
        cout << endl;
    }

    for (Creature* creature : zoo) {
       delete creature;
    }

    return 0;
}
```