**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Программирование»**

Тема: **Уровни абстракции**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2381 |  | Рыжиков И.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

## Цель работы

Целью работы является изучение полиморфизма и взаимосвязей между классами, которые реализуют один и тот же интерфейс в языке C++.

## Задание

а) Создать интерфейс игрового события. Интерфейс должен обеспечивать срабатывание события когда игрок наступает на клетку.

б) Реализовать интерфейс игрового события тремя конкретными событиями. Одно событие должно положительно влиять на характеристики игрока, второе должно негативно влиять на характеристики игрока, третье изменять координаты игрока на поле. При желании можно реализовать больше событий и/или события меняющие само поле (например, делать из непроходимой клетки проходимую).

в) В классе управления игроком добавить проверку на наличие события на клетке, если событие присутствует, то оно должно сработать. Срабатывание должно происходить через интерфейс события, и не должно быть никаких проверок на тип события (реализация через динамический полиморфизм)

г) Создать класс создающий поле. Предусмотреть возможность создания 2 разных уровней. По желанию можно сделать случайную генерацию уровней. Должно гарантироваться, что игрок может дойти от входа до выхода.

Примечания:

* События должны быть такими, чтобы был сценарий проигрыша игрока.
* В событиях и клетках не должно быть полей сообщающих информацию о типе события

## Выполнение работы

### Класс FileReader

Класс FileReader предоставляет базовую функциональность для чтения файла в C++. Вот краткое описание его методов:

1. FileReader::FileReader(const std::string &fileName)**:**
   * Конструктор класса FileReader.
   * Принимает параметр fileName - имя файла для чтения.
   * Открывает файл в бинарном режиме для чтения (std::ios::in | std::ios::binary).
   * Если файл не удается открыть, генерирует исключение std::runtime\_error с сообщением об ошибке.
2. FileReader::~FileReader()**:**
   * Деструктор класса FileReader.
   * Закрывает файл, если он был открыт.
3. bool FileReader::readLine(std::string &line)**:**
   * Метод для чтения строки из файла.
   * Принимает ссылку на строку (line), в которую будет записана прочитанная строка.
   * Возвращает true, если строка успешно прочитана из файла.
   * Возвращает false, если достигнут конец файла или произошла ошибка при чтении.

### Класс KeyboardInput

Класс KeyboardInput предоставляет функциональность для чтения команд с клавиатуры и создания соответствия клавишам и управляющим командам. Вот краткое описание его методов:

1. KeyboardInput::KeyboardInput(const std::string &filename)**:**
   * Конструктор класса KeyboardInput.
   * Принимает параметр filename - имя файла с соответствием клавишам и командам.
   * Вызывает метод createKeyboardLayoutMap(filename) для создания карты клавиш.

Controls KeyboardInput::read():

* + Метод для чтения команды с клавиатуры.
  + Использует функцию getch() для считывания кода клавиши.
  + Возвращает соответствующую команду из созданной карты клавиш.

1. KeyboardInput::createKeyboardLayoutMap(const std::string &filename)**:**
   * Статический метод класса KeyboardInput для создания карты клавиш из файла.
   * Принимает параметр filename - имя файла с соответствием клавишам и командам.
   * Использует FileReader для чтения файла.
   * Создает карту клавиш и соответствующих им команд.
   * Проверяет уникальность значений и наличие команд в файле, генерируя исключения при несоответствиях.

### Перечисление Controls

Заголовочный файл Controls.h содержит объявление перечисления Controls, представляющего возможные управляющие команды. Вот краткое описание его содержимого:

1. enum Controls { QUIT = -1, MOVE\_UP = 0, MOVE\_RIGHT = 1, MOVE\_DOWN = 2, MOVE\_LEFT = 3 };
   * О пределяет перечень управляющих команд.
   * **QUIT**: Команда завершения программы (может быть использована как специальное значение для выхода).
   * **MOVE\_UP**, **MOVE\_RIGHT**, **MOVE\_DOWN**, **MOVE\_LEFT**: Команды движения в различных направлениях.

### Класс Game

Класс Game представляет собой реализацию игры с использованием различных компонентов, таких как игровое поле (Field), игрок (Player), контроллер (PlayerController), вид (FieldView), и ввод с клавиатуры (KeyboardInput). Вот краткое описание его методов:

1. **Конструктор:**
   * Game::Game():
     + Конструктор класса **Game**.
     + Инициализирует уровень, флаги состояний (**win** и **loose**), вид, контроллер и представление поля.
2. **Методы:**
   * void Game::switchLevel(int level\_):
     + Меняет текущий уровень на указанный level\_.
     + Создает новое поле с использованием объекта FieldCreator.
     + Обновляет контроллер, связанный с новым полем.
   * void Game::startGame():
     + Запускает игру.
     + Вызывает метод управления игроком (runControlPlayer()) и завершает игру после его выполнения (endGame()).
   * void Game::runControlPlayer():
     + Метод для обработки ввода от игрока с клавиатуры.
     + Использует объект KeyboardInput для чтения команд от игрока.
     + Обновляет состояние игры в соответствии с командами.
   * bool Game::isLoose()**:**
     + Проверяет, проиграл ли игрок.
     + Возвращает true, если игрок мертв.
   * bool Game::isWin()**:**
     + Проверяет, выиграл ли игрок.
     + Возвращает true, если игрок достиг финиша на поле.
   * void Game::endGame() const**:**
     + Выводит сообщение о завершении игры в зависимости от результата (победа, поражение).

### Класс GameMenu

Класс GameMenu предоставляет простое текстовое меню для взаимодействия с игрой. Вот краткое описание его методов:

1. **Метод:**
   * void GameMenu::start()**:**
     + Начинает игровое меню.
     + Запрашивает у пользователя выбор уровня для запуска игры.
     + После выбора уровня, вызывает методы switchLevel и startGame класса Game.
     + Повторяет процесс выбора уровня, пока пользователь не введет "q" для выхода.

### Сктруктура Coordinate

Структура Coordinate была дополнена оператором равенства (==), который сравнивает координаты двух объектов Coordinate. Вот краткое описание внесенных изменений:

1. **Оператор равенства:**
   * bool operator==(const Coordinate &other) const**:**
     + Оператор сравнения равенства для структуры Coordinate.
     + Сравнивает координаты (x и y) текущего объекта с координатами другого объекта Coordinate.
     + Возвращает true, если координаты равны, и false в противном случае.

## Выводы

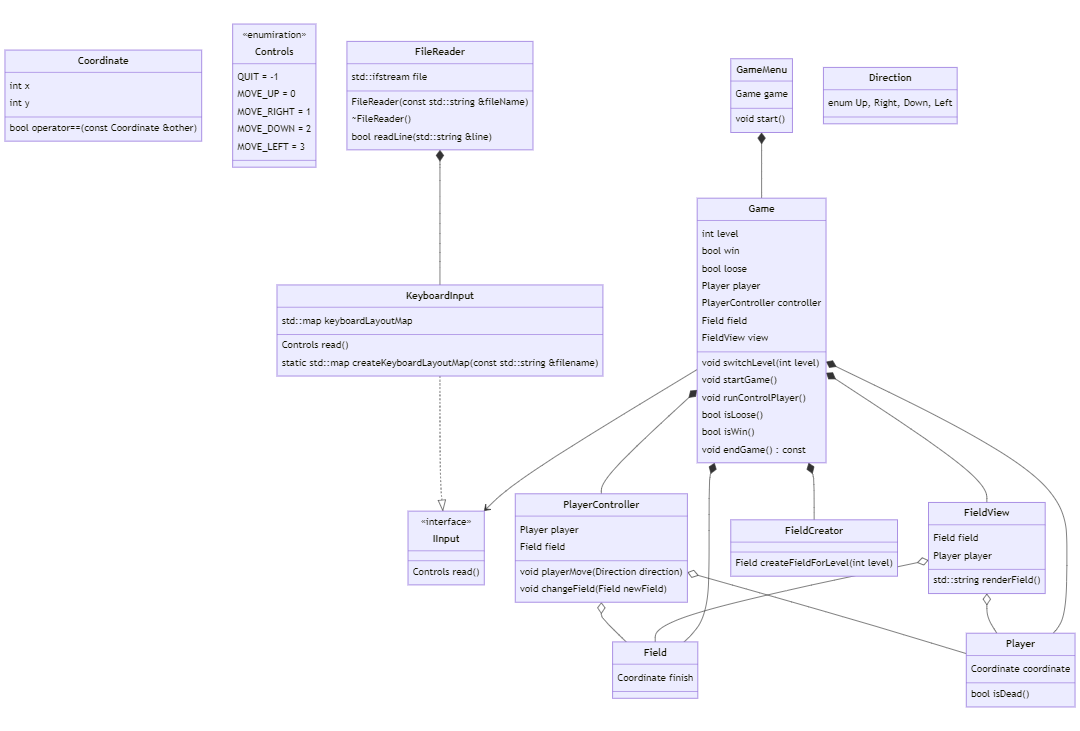
Произведенная работа включает создание двух важных компонентов для игрового приложения: класса игры (Game) и класса считывания ввода пользователя с клавиатуры (KeyboardInput). Оба класса написаны с использованием современных подходов к программированию на C++ и включают в себя следующие ключевые аспекты:

1. **Класс Game:**
   * Реализует логику игры с использованием компонентов, таких как игровое поле, игрок, контроллер и представление поля.
   * Работает с вводом пользователя с клавиатуры через класс KeyboardInput.
   * Реализует методы для управления игрой, изменения уровня, обработки ввода пользователя и определения состояния победы/поражения.
   * Применяется идиома RAII при работе с файлом в классе FileReader.
2. **Класс KeyboardInput:**
   * Реализует интерфейс IInput, который включает в себя чисто виртуальный метод read.
   * Осуществляет чтение команд пользователя с клавиатуры и их преобразование в объекты команд (Controls).
   * Включает в себя статический метод для создания карты клавиш и соответствующих им команд из файла.
   * Генерирует исключение при ошибке открытия файла, что обеспечивает безопасное и контролируемое взаимодействие с внешними ресурсами.

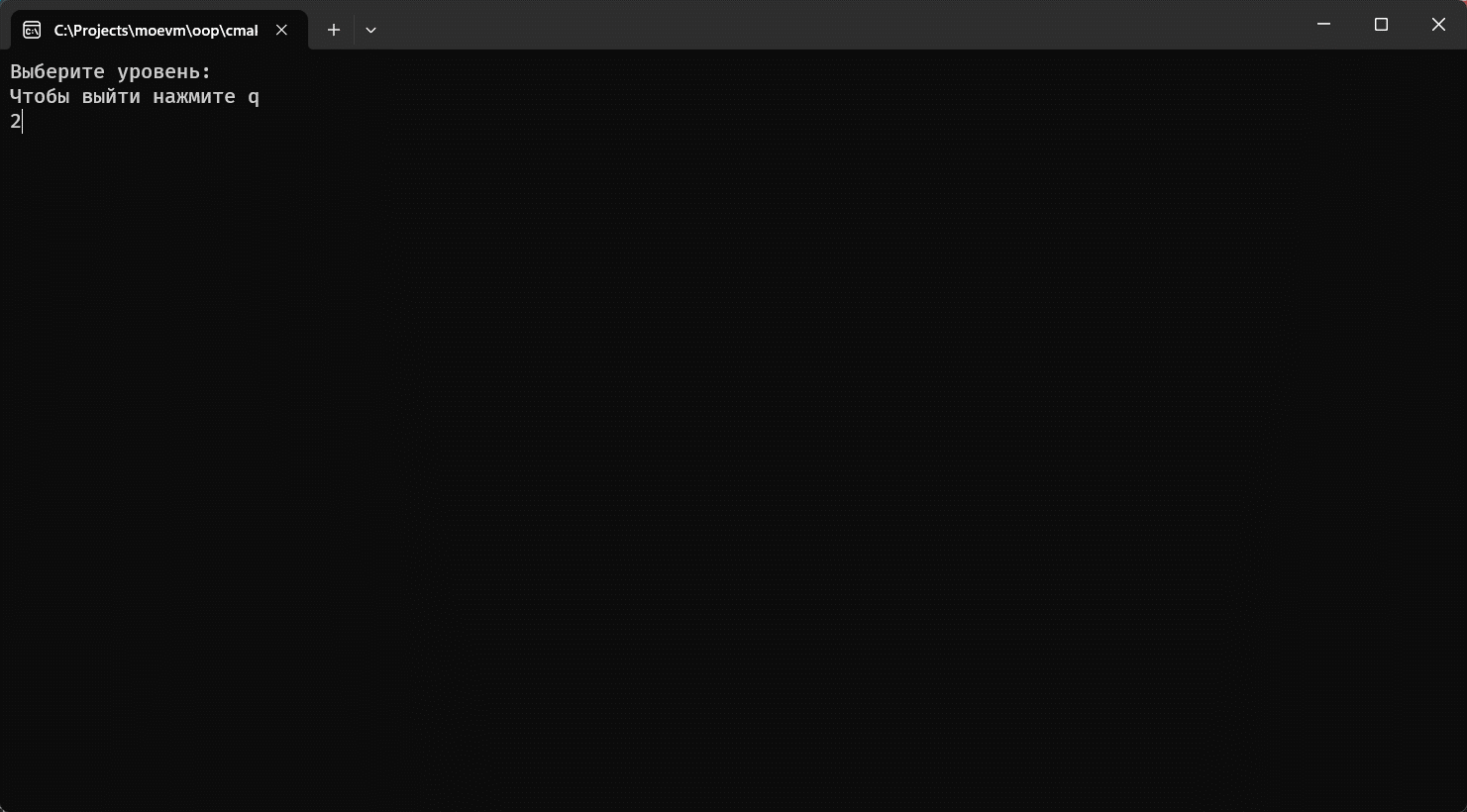
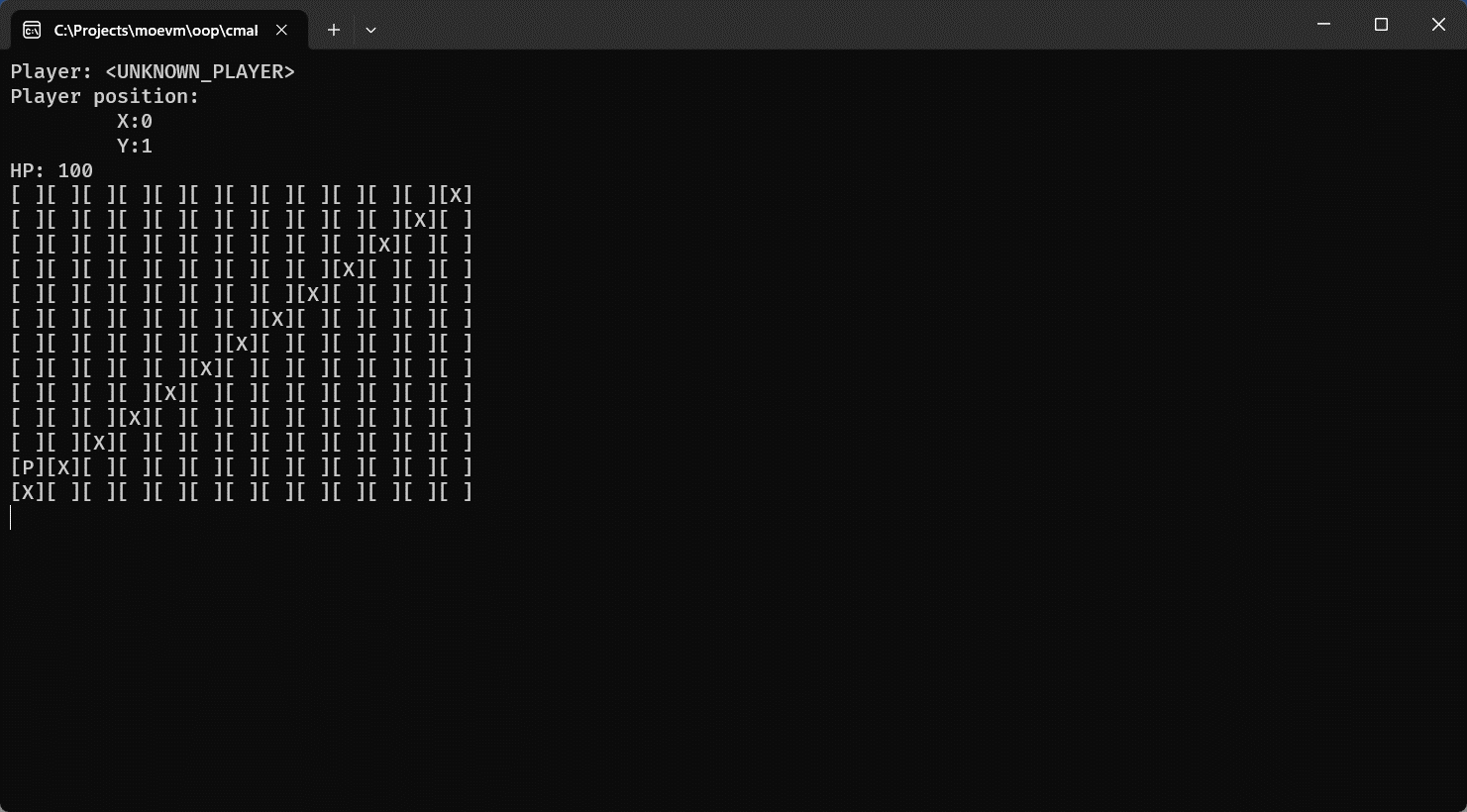
Таким образом, созданные классы обеспечивают структурированную и модульную архитектуру для разработки игрового приложения. Ввод с клавиатуры обрабатывается немедленно, и дальнейшая работа проводится с объектами, представляющими команды, что способствует легкости поддержки и расширения приложения. Использование идиомы RAII при работе с файлом обеспечивает автоматическое управление ресурсами и предотвращает утечки памяти.

Пример работы программы (рис. 1) и диаграмма классов (рис. 2) представлена в Приложении.

# Приложение



1. Диаграмма классов

1. Пример работы программы