МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ №3

по лабораторной работе

по дисциплине «Объектно-ориентированное программирование»

Тема: Логирование, перегрузка операторов

Студент гр. 1381 _____ Дудко М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Ознакомиться с работой основных принципов ООП. Написать примитивную игру на языке C++.

Задание.

Реализовать класс/набор классов отслеживающих изменения состояний в программе. Отслеживание должно быть 3-х уровней:

1. Изменения состояния игрока и поля, а также срабатывание событий
2. Состояние игры (игра начата, завершена, сохранена, и.т.д.)
3. Отслеживание критических состояний и ошибок (поле инициализировано с отрицательными размерами, игрок попытался перейти на непроходимую клетку, и.т.д.)

Реализованы классы для вывода информации разных уровней для в консоль и в файл с перегруженным оператором вывода в поток.

Выполнение работы.

В классе Game_Changes были сохранены сообщения лога в виде полей класса, а именно префикс, текущее время и текст. В методах этого класса я инициализирую эти поля, получая значение из других классов, или прописывая их самостоятельно.

```
void Game_Changes::Field_Message(Field& field) {
    std::string get_player_position_x = std::to_string( val: field.get_player_position_x());
    std::string get_player_position_y = std::to_string( val: field.get_player_position_y());
    std::string s1 = "get_player_position_x: ";
    std::string s2 = "get_player_position_y: ";
    message = s1 + get_player_position_x + " " + s2 + get_player_position_y + " ";
}

void Game_Changes::Event_Message(Controller_p& controllerP){
    if(controllerP.get_is_event()) message = "Is event!";
    else message = "None";
}

void Game_Changes::Create_Message(std::string c_message){
    message = c_message;

}
```

Рисунок 1 Инициализация полей

Так же был перегружен оператор ввода, теперь получая на вход объект класса Game_Changes он будет выводить полное сообщение лога включающее текущее время, префикс и текст

```
std::ostream& operator<<(std::ostream& out, const Game_Changes& obj){
    out << obj.time_cur;
    out << " ";
    out << obj.prefix;
    out << " ";
    out << obj.message << std::endl;
    return out;
}
```

Рисунок 2 Перегрузка оператора вывода

Был разработан общий интерфейс и создана виртуальный метод print для переопределения в дочерних классов для разных форматов вывода

```
class Interface_output{
public:
    virtual ~Interface_output()=default;
    virtual void print(const Game_Changes& obj)=0;

};
```

Рисунок 3 Интерфейс

Класс File_Output ведет вывод в файл. В нем есть конструктор (открывает файл), деструктор (закрывает файл), а так же переопределенный метод интерфейса – вывод в файл.

```
File_Output::File_Output(){
    file.open( s: "log.txt");
};

File_Output::~File_Output(){
    file.close();
}

void File_Output::print(const Game_Changes& obj){
    file << obj;
}
```

Рисунок 4 File_Output

В классе Console_Output метод переопределен уже для вывода в консоль.

В классе Logger были созданы методы логов, которые инициализирует поля объекта Game_Changes, а так же уровень логирования исходя из сути самого лога, а затем производится вывод с помощью внутреннего метода log, который проверяет уровень логирования, а так же способ вывода логов, уровень логирования реализован с помощью перечисления enum. Метод switch_level сохраняет в переменную элемент списка enum. Выбор способов вывода реализован в методе switch_out посредством оператора switch. А так же смены состояния переменных типа Boolean.

В main на различных этапах игры вызываются методы вывода логов. И в начале выбора уровня логирования, а так же способов вывода

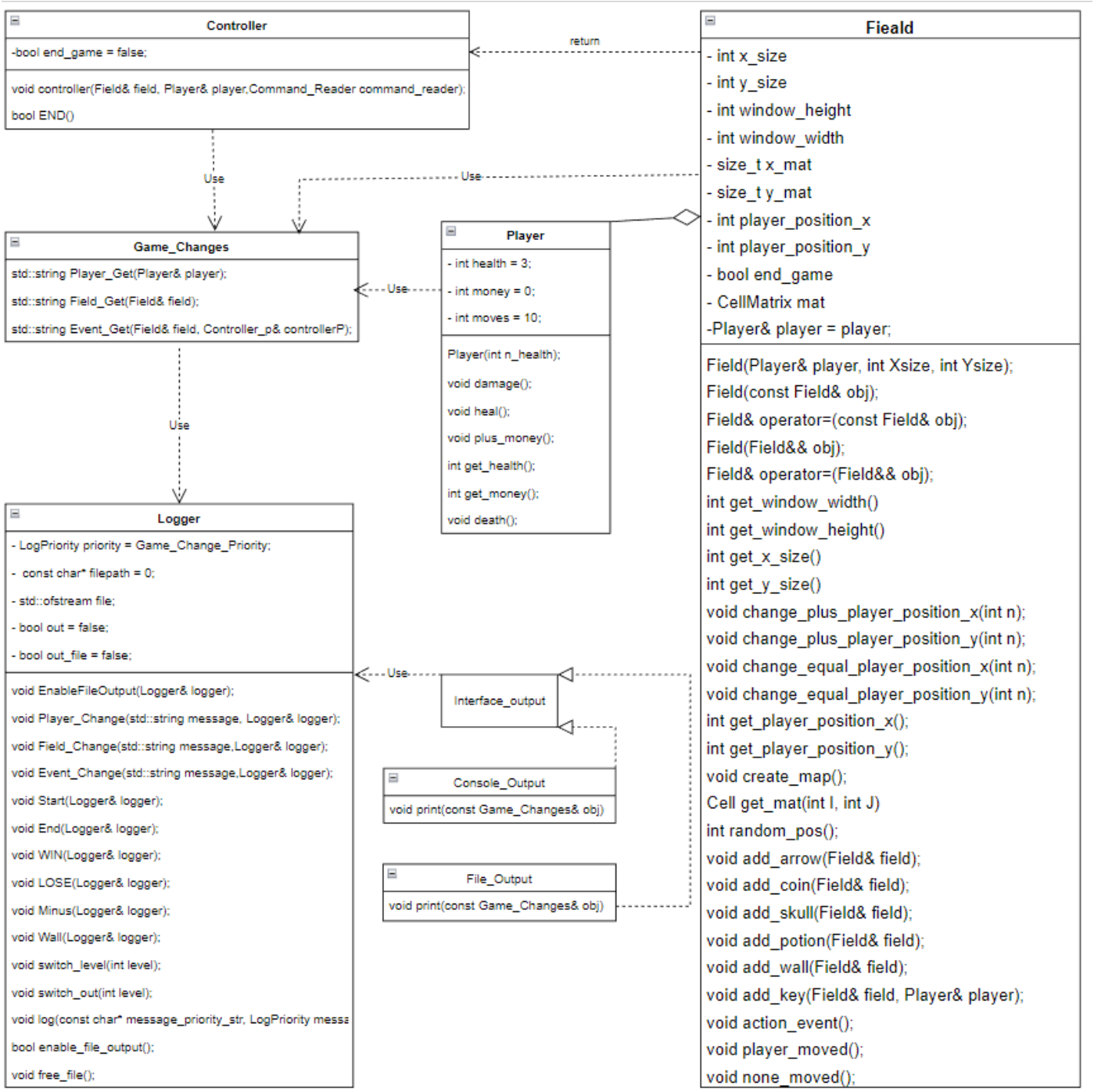

Рисунок 5 ЮМЛ ДИАГРАММА