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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Объектно-ориентированное программирование»
Тема: Сохранение и загрузка / Написание исключений

Студент гр. 9382 Демин В.В.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Создать класс сохранения и загрузки.

Задание.

Создать классы, которые позволяют сохранить игру, а потом загрузить ее. Также, написать набор исключений, которые как минимум позволяют контролировать процесс сохранения и загрузки

Обязательные требования:

- Игру можно сохранить в файл
- Игру можно загрузить из файла
- Взаимодействие с файлами по идиоме RAII
- Добавлена проверка файлов на корректность
- Написаны исключения, которые обеспечивают транзакционность

Дополнительные требования:

- Для получения состояния программы используется паттерн Снимок

Выполнение работы.

Был создан класс Save_and_Load, который и выполняет сохранение и загрузку игры. С помощью паттерна снимок этот класс получает поля, которые необходимо записать в файл. Записываются они бинарно. При загрузки сначала происходит проверка файла на корректность и если файл прошел проверку, то происходит загрузка сохранения в игру.

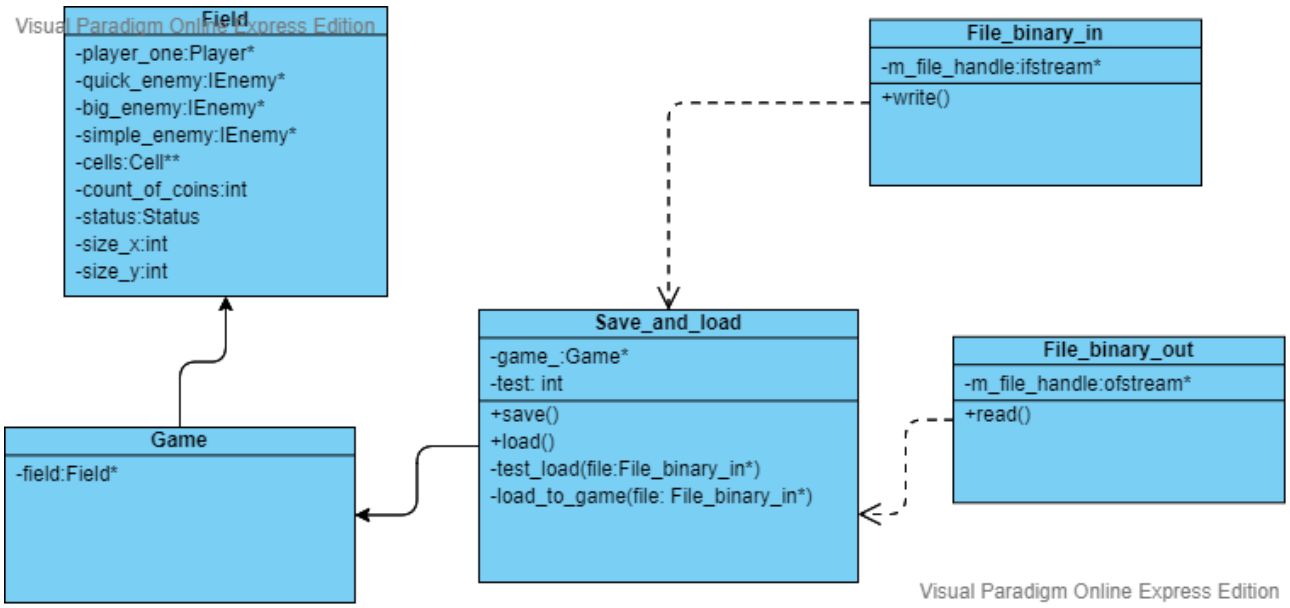

Рисунок 1. Uml диаграмма

Выводы.

В ходе выполнения лабораторной работы был создан класс для сохранения и загрузки игры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: GameAndLoad.h

```
//
// Created by vikto on 24.11.2020.
//

#ifndef MYGAME_SAVEANDLOAD_H
#define MYGAME_SAVEANDLOAD_H

#include "../../Game/Game.h"
#include "fstream"
#include "../Errors/InvalidFileException.h"

class File_binary_out;

class File_binary_in;

class SaveAndLoad {
    Game *game_;
    int test;

    void test_load(File_binary_in *file);

    void load_to_game(File_binary_in *file);

public:
    SaveAndLoad(Game *game);

    void save();


    void load();
};

class File_binary_out {
public:
    File_binary_out(const char *filename) {
        m_file_handle = new std::ofstream(filename, std::ios::binary);
        if (!m_file_handle->is_open())
            throw std::runtime_error("file open failure");
    }

    ~File_binary_out() {
```

```
        m_file_handle->close();
    }

    void write(const int &a) {
        std::cout << a;
        m_file_handle->write((char *) &a, sizeof(a));
        if (!m_file_handle)
            throw std::runtime_error("file write failure");
    }


private:
    std::ofstream *m_file_handle;

    File_binary_out(const File_binary_out &);

    File_binary_out &operator=(const File_binary_out &);
};

class File_binary_in {
public:
    File_binary_in(const char *filename) {
        m_file_handle = new std::ifstream(filename, std::ios::binary);
        if (!m_file_handle->is_open())
            throw std::runtime_error("file open failure");
    }

    ~File_binary_in() {
        m_file_handle->close();
    }

    int read() {

        m_file_handle->read((char *) &mem, sizeof(mem));
        std::cout << mem;
        return mem;
    }

    void begin() {

        m_file_handle->clear();
        m_file_handle->seekg(std::ios::beg);
    }
```

```
private:
    std::ifstream *m_file_handle;
    int mem = 0;
    Element *elem;
    int tell;

    File_binary_in(const File_binary_in &);

    File_binary_in &operator=(const File_binary_in &);
};


#endif //MYGAME_SAVEANDLOAD_H
```

Название файла: GameAndLoad.cpp

```
//
// Created by vikto on 24.11.2020.
//

#include "SaveAndLoad.h"

void SaveAndLoad::save() {
    std::string f_name = "Save/save.txt";
    Game::Memento mem = game_->save_memento();
    File_binary_out *file = new File_binary_out(f_name.c_str());

    file->write(mem.field->get_size_x());
    file->write(mem.field->get_size_y());

    file->write(mem.field->get_player()->get_cord_x());
    file->write(mem.field->get_player()->get_cord_y());
    file->write(mem.field->get_player()->get_coins_player());
    file->write(mem.field->get_player()->get_health_player());

    if (mem.field->get_big_enemy()) {
        file->write(1);
        file->write(mem.field->get_big_enemy()->get_cord_x());
        file->write(mem.field->get_big_enemy()->get_cord_y());
    } else {
        file->write(0);
    }
    if (mem.field->get_simple_enemy()) {
        file->write(1);
```

```
        file->write(mem.field->get_simple_enemy()->get_cord_x());
        file->write(mem.field->get_simple_enemy()->get_cord_y());
    } else {
        file->write(0);
    }
    if (mem.field->get_quick_enemy()) {
        file->write(1);
        file->write(mem.field->get_quick_enemy()->get_cord_y());
        file->write(mem.field->get_quick_enemy()->get_cord_y());
    } else {
        file->write(0);
    }
    file->write(mem.field->get_count_coins());
    file->write(mem.field->get_status());
    Cell **write_cells = mem.field->get_cells();
    for (int i = 0; i < mem.field->get_size_y(); ++i) {
        for (int j = 0; j < mem.field->get_size_x(); ++j) {
            file->write(write_cells[i][j].get_type());
            if (write_cells[i][j].get_element()) {
                file->write(1);
                file->write(write_cells[i][j].get_element()->get_coin());
                file->write(write_cells[i][j].get_element()->get_health());
                file->write(write_cells[i][j].get_element()->get_status());
            } else {
                file->write(0);
            }
        }
    }
    getch();
    delete file;
}

void SaveAndLoad::load() {
    std::string f_name = "Save/save.txt";
    try {
        File_binary_in *file = new File_binary_in(f_name.c_str());
        test_load(file);
        file->begin();
        load_to_game(file);
    } catch (InvalidFileException exception) {
        std::cout << exception.invalid_file();
        getch();
        return;
    }
}
```

```
SaveAndLoad::SaveAndLoad(Game *game) : game_(game) {

}

void SaveAndLoad::test_load(File_binary_in *file) {
    int byte = 0;
    int test_x = file->read();
    if (test_x < 0 || test_x > 150) {
        throw InvalidFileException(byte);
    }
    byte++;
    int test_y = file->read();

    if (test_y < 0 || test_y > 150) {
        throw InvalidFileException(byte);
    }
    byte++;
    test = file->read();
    if (test > test_x - 1 || test < 1) {
        throw InvalidFileException(byte);
    }
    byte++;
    test = file->read();
    if (test > test_y - 1 || test < 1) {
        throw InvalidFileException(byte);
    }
    byte++;
    test = file->read();
    if (test > test_x * test_y / 50 || test < 0) {
        throw InvalidFileException(byte);
    }
    byte++;
    test = file->read();
    if (test < 0) {
        throw InvalidFileException(byte);
    }
    byte++;
    for (int i = 0; i < 3; ++i) {
        test = file->read();
        if (test != 0 && test != 1) {
            throw InvalidFileException(byte);
        }
        byte++;
        test = file->read();
```

```
    if (test > test_x - 1 || test < 1) {
        throw InvalidFileException(byte);
    }
    byte++;
    test = file->read();
    if (test > test_y - 1 || test < 1) {
        throw InvalidFileException(byte);
    }
    byte++;
}
test = file->read();
if (test > test_x * test_y / 50 || test < 0) {
    throw InvalidFileException(byte);
}
byte++;
test = file->read();
if (test < 0 || test > 5) {
    throw InvalidFileException(byte);
}
byte++;
for (int i = 0; i < test_y; ++i) {
    for (int j = 0; j < test_x; ++j) {
        test = file->read();
        if (test < 0 || test > 5) {
            throw InvalidFileException(byte);
        }
        byte++;
        test = file->read();
        byte++;
        if (test != 0 && test != 1) {
            throw InvalidFileException(byte);
        }
        if (test) {
            for (int k = 0; k < 2; ++k) {
                test = file->read();
                if (test < 0) {
                    throw InvalidFileException(byte);
                }
                byte++;
            }
            test = file->read();
            if ((test < 0 || test > 5)) {
                throw InvalidFileException(byte);
            }
            byte++;
```

```
            }
        }
    }
}

void SaveAndLoad::load_to_game(File_binary_in *file) {
    Element *elem;
    int coin = 0;
    int health = 0;
    Status status;

    Field *field = Field::get_instance();
    field->set_size(file->read(), file->read());
    field->get_player()->set_cord_x(file->read());
    field->get_player()->set_cord_y(file->read());
    field->get_player()->set_coins_player(file->read());
    field->get_player()->set_health_player(file->read());
    if (file->read()) {
        field->get_big_enemy()->set_cord_x(file->read());
        field->get_big_enemy()->set_cord_y(file->read());
    }
    if (file->read()) {
        field->get_simple_enemy()->set_cord_x(file->read());
        field->get_simple_enemy()->set_cord_y(file->read());
    }
    if (file->read()) {
        field->get_quick_enemy()->set_cord_x(file->read());
        field->get_quick_enemy()->set_cord_y(file->read());
    }
    field->set_count_coint(file->read());
    field->set_status((Status) file->read());
    Cell **cells = new Cell *[field->get_size_y()];
    for (int i = 0; i < field->get_size_y(); ++i) {
        cells[i] = new Cell[field->get_size_x()];
        for (int j = 0; j < field->get_size_x(); ++j) {
            cells[i][j].set_type((TypeCell) file->read());
            if (cells[i][j].get_type() != EMPTY) {
                file->read();
                continue;
            }
            if (file->read()) {
                coin = file->read();
                health = file->read();
                status = (Status) file->read();
                if (coin == 1) {
```

```
                elem = new Coin();
            } else if (health == 1) {
                elem = new Mushroom();
            } else {
                elem = new Chest();
                elem->set_coin(coin);
                elem->set_health(health);
                elem->set_status(status);
            }
            cells[i][j].set_element(elem);
        }
    }
  }
  field->set_cells(cells);
  delete file;
}
```