**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.

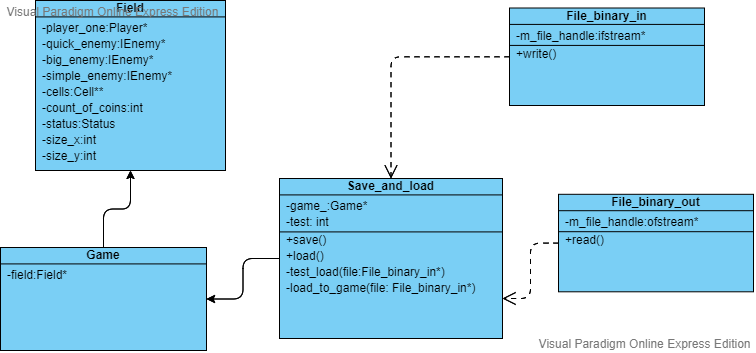


Рисунок 1. Uml диаграмма

## 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;

}