**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Логирование, перегрузка операций

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Парамонов В.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

## Цель работы.

Реализовать возможность логирования состояния и изменения различных объектов игры в ходе работы с возможностью вывода в файл или консоль, а также изменения формата логов.

## Задание.

Необходимо проводить логирование того, что происходит во время игры.

Требования:

* Реализован класс логгера, который будет получать объект, который необходимо отслеживать, и при изменении его состоянии записывать данную информацию.
* Должна быть возможность записывания логов в файл, в консоль или одновременно в файл и консоль.
* Должна быть возможность выбрать типа вывода логов
* Все объекты должны логироваться через перегруженный оператор вывода в поток.
* Должна соблюдаться идиома RAII.

**Основные теоретические положения.**

Потенциальные паттерны проектирования, которые можно использовать:

* Адаптер (*Adapter*) - преобразование данных к нужному формату логирования.
* Декоратор (*Decorator*) - форматирование текста для логирования.
* Мост (*Bridge*) - переключение между логированием в файл/консоль.
* Наблюдатель (*Observer*) - отслеживание объектов, которые необходимо логировать.
* Синглтон (*Singleton*) - гарантия логирования в одно место через одну сущность.
* Заместитель (*Proxy*) - подстановка и выбор необходимого логирования.

## Выполнение работы.

1. Логирование:

* Класс *Log* – обертка для сообщения, содержащий само сообщение в *log\_message\_* и информацию о выбранном стиле (выбранных стилях) в *log\_style\_*. Имеет два геттера *getMessage* и *getStyle*для получения сообщения или стиля соответственно.
* Классы *LogConsole* и *LogFile* – наследники класса , необходимы для определения внутри класса-логгера в какой поток следует выводить данное сообщение (в консоль или в файл соответственно).
* Перед рассмотрением двух основных классов *Logger* и *Loggable* остановимся на двух вспомогательных перечислениях: *LogMode*, который нужен для выбора вывода в консоль или в файл (*kConsole* и *kFile*), и *LogStyle*, позволяющий выбрать различные оформления логов. Оба перечисления созданы таким образом, чтобы через оператор “или” была возможность выбрать несколько вариантов. Таким образом можно выбрать вывод в файл и в консоль одновременно или несколько стилей.
* Теперь рассмотрим класс *Logger* – сам логгер. Он создан с использованием Паттерна “одиночка” и при создании объекта открывает поток в файл по стандартному пути (*log\_path\_* = *"log.txt"*). Изменить путь можно с помощью метода *setLogPath*. Вывод логов осуществляется через 2 перегруженных оператора вывода: Logger& operator << (LogFile& log\_info) и Logger& operator << (LogConsole& log\_info). В зависимости от вида класса логов используется первую (для вывода в файл) или вторую (для вывода в консоль) перегрузку. Перед выводом логи стилизуются в соответствии c выбранными в них *LogStyle* методом *stylize\_log*.
* И, наконец, последнее звено механизма логирования - класс *Loggable*, позволяющий логировать унаследованные от него классы. Логирование включается с помощью метода *LogActive,* который ставит внутренний флаг *logging\_active\_* в значение *true*, выключается с помощью *LogDisable,* который ставит *logging\_active\_* в значение *false*. Сама информация в логгер передается с помощью метода *LogInfo*, который принимает: куда мы выводим логи (*LogMode*), стиль логов (*LogStyle*) и само сообщение *message*, после этого в зависимости от выбранного вывода, переданное сообщение или стиль оборачивается в класс *LogConsole* или *LogFile*, эта обертка передается логгеру через оператор вывода.

1. Бизнес-логика:

* Класс *GameLogic,*  олицетворяющий связь бизнес-логики игры, стал наследоваться от *Loggable* для поддержки логирования. Здесь логируются события создания комнаты с выводом её размера, события создания всех объектов комнаты и их характеристики, взаимодействия объектов друг с другом в ходе игры.

1. GUI:

* Класс *GameView*– основа графического интерфейса, стал наследоваться от *Loggable* для поддержки его логирования. В целях теста информация из этого класса логируется только про изменение размера экрана приложения.

Разработанный программный код см. в приложении А.

Тестирование см. в приложении Б.

UML-диаграмму см. в приложении В.

## Выводы.

Был реализован вывод логов во время работы приложения в консоль и в файл, для логирования объекта достаточно унаследовать его от *Loggable*, добавить, где необходимо получать информацию, *LogInfo*, и установить объект логируемым через метод *LogActive*.

# Приложение А Исходный код программы

Название файла: main.cpp

#include <QApplication>

#include "Gui/GameView.h"

#include "Logic/Room/Room.h"

#include "Logic/Room/Pos.h"

#include "Logic/GameLogic.h"

#include "Logic/Controller/Controller.h"

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

GameLogic logic;

logic.LogActive();

GameView game\_view(*&logic*);

game\_view.LogActive();

logic.CreateRoom();

Controller control(*&logic*, *&game\_view*);

game\_view.show();

return a.exec();

}

Название файла: Log.h

#ifndef LOG\_H

#define LOG\_H

#include "string"

#include "../UtilityEnums/LogStyle.h"

class **Log** {

private:

std::string log\_message;

int log\_style;

public:

**Log**(std::string message, LogStyle style);

std::string **getMessage**();

int **getStyle**();

};

#endif // LOG\_H

Название файла: LogConsole.h

#ifndef LOGCONSOLE\_H

#define LOGCONSOLE\_H

#include "Log.h"

class **LogConsole**: public Log {

public:

**LogConsole**(std::string message, LogStyle style);

};

#endif // LOGCONSOLE\_H

Название файла: LogFile.h

#ifndef LOGFILE\_H

#define LOGFILE\_H

#include "Log.h"

class **LogFile**: public Log {

public:

**LogFile**(std::string message, LogStyle style);

};

#endif // LOGFILE\_H

Название файла: Loggable.h

#ifndef LOGGABLE\_H

#define LOGGABLE\_H

#include "string"

#include "../UtilityEnums/LogMode.h"

#include "../UtilityEnums/LogStyle.h"

#include "../Logger/Logger.h"

#include "../Log/LogFile.h"

#include "../Log/LogConsole.h"

class Loggable {

bool logging\_active = false;

public:

void LogActive();

void LogDisable();

void LogInfo(int mode, int style, std::string message);

};

#endif // LOGGABLE\_H

Название файла: Logger.h

#ifndef LOGGER\_H

#define LOGGER\_H

#include <time.h>

#include <iostream>

#include <stdexcept>

#include <fstream>

#include <string>

#include "../Log/LogFile.h"

#include "../Log/LogConsole.h"

#include "../UtilityEnums/LogStyle.h"

class **Logger** {

private:

static Logger\* logger\_inst;

std::ofstream file\_stream;

std::string log\_path;

long long int log\_count;

**Logger**();

public:

static Logger& **getInstance**();

**Logger**(const Logger& from) = delete;

Logger& operator =(const Logger& from) = delete;

void **setLogPath**(std::string new\_path);

Logger& operator << (LogFile& log\_info);

Logger& operator << (LogConsole& log\_info);

std::string **stylize\_log**(Log& log\_info);

~**Logger**();

};

#endif // LOGGER\_H

Название файла: LogMode.h

#ifndef LOGMODE\_H

#define LOGMODE\_H

enum **LogMode** {

kConsole = 0b01,

kFile = 0b10

};

#endif // LOGMODE\_H

Название файла: LogStyle.h

#ifndef LOGSTYLE\_H

#define LOGSTYLE\_H

enum **LogStyle** {

kCapitalise = 0b001,

kTime = 0b010,

kNumbered = 0b100

};

#endif // LOGSTYLE\_H

Название файла: GameLogic.h

#ifndef GAMELOGIC\_H

#define GAMELOGIC\_H

#include <vector>

#include <QFile>

#include "../Gui/View/Observable.h"

#include"../Logging/Loggable/Loggable.h"

#include "Room/Room.h"

#include "Room/Pos.h"

#include "PlaceableInCell/Player/Player.h"

#include "PlaceableInCell/Enemies/DamageableEnemy.h"

#include "PlaceableInCell/Enemies/LimitedLifeEnemy.h"

#include "PlaceableInCell/Enemies/ImmortalEnemy.h"

#include "PlaceableInCell/Items/SanityAffectItem.h"

#include "PlaceableInCell/Items/DamageToEnemiesItem.h"

#include "PlaceableInCell/Items/KeyItem.h"

#include "PlaceableInCell/Enemies/MoveStrategies/StrategiesEnum.h"

class **GameLogic**: public Observable, public Loggable {

private:

Room\* curr\_room\_;

std::vector<PlaceableInCell\*> room\_objects\_;

std::map<MoveStrategies, MoveStrategy\*> strategy\_list;

public:

**GameLogic**();

void **CreateRoom**(int id = 1, int is\_new\_or\_load = 0);

void **Start**();

Room& **getRoom**();

std::vector<PlaceableInCell\*>& **getRoomObjects**();

void **LoadCellObjects**(QFile& obj\_file);

void **MakeTurn**(Pos player\_pos\_change);

~**GameLogic**();

};

#endif // GAMELOGIC\_H

Название файла: GameView.h

#ifndef GAMEVIEW\_H

#define GAMEVIEW\_H

#include <vector>

#include <typeinfo>

#include <QSizePolicy>

#include <QResizeEvent>

#include <QGraphicsScene>

#include <QGraphicsPixmapItem>

#include <QPropertyAnimation>

#include <QLabel>

#include <QWidget>

#include "View/Observer.h"

#include "../Logging/Loggable/Loggable.h"

#include "../Logic/GameLogic.h"

#include "DrawObject/DrawableObject.h"

namespace **Ui** {

class **GameView**;

}

class **GameView** : public QWidget, public Observer, public Loggable {

private:

Q\_OBJECT

Ui::GameView \*ui;

QGraphicsScene\* room\_scene;

GameLogic\* curr\_logic\_;

std::vector<DrawableObject\*> draw\_objects\_;

int floor\_cell\_size\_ = 500;

int top\_padding\_scene\_ = 500;

virtual void ***resizeEvent***(QResizeEvent\* event);

virtual void ***keyPressEvent***(QKeyEvent \*event);

QLabel\* test\_info;

public:

explicit **GameView**(GameLogic\* new\_logic, QWidget \*parent = nullptr);

void ***UpdateRoomChange***();

void ***UpdateRoomObjectsChange***();

void ***UpdateObjectDestroy***(int i);

void ***UpdateTurnComplete***();

void ***UpdateWin***();

void ***UpdateDefeat***();

void **TestShowPlayerConditions**();

void **CenterViewOnPlayerPos**();

void **ClearObjects**();

void **DeleteRoom**();

~***GameView***();

signals:

void **PlayerPressMove**(Pos);

};

#endif // GAMEVIEW\_H

Название файла: Log.cpp

#include "Log.h"

Log::**Log**(std::string message, LogStyle style) {

log\_message = message;

log\_style = style;

}

std::string Log::**getMessage**() {

return log\_message;

}

int Log::**getStyle**() {

return log\_style;

}

Название файла: LogConsole.cpp

#include "LogConsole.h"

LogConsole::**LogConsole**(std::string message, LogStyle style): Log(message, style) {}

Название файла: LogFile.cpp

#include "LogFile.h"

LogFile::**LogFile**(std::string message, LogStyle style): Log(message, style) {}

Название файла: Loggable.cpp

#include "Loggable.h"

void Loggable::**LogActive**() {

logging\_active = true;

}

void Loggable::**LogDisable**() {

logging\_active = false;

}

void Loggable::**LogInfo**(int mode, int style, std::string message) {

if (logging\_active) {

if (mode & LogMode::kFile) {

LogFile log\_file = LogFile(message, static\_cast<LogStyle>(style));

Logger::getInstance() << log\_file;

}

if (mode & LogMode::kConsole) {

LogConsole log\_console = LogConsole(message, static\_cast<LogStyle>(style));

Logger::getInstance() << log\_console;

}

}

}

Название файла: Logger.cpp

#include "Logger.h"

Logger\* Logger::logger\_inst = nullptr;

Logger::**Logger**() {

log\_path = "log.txt";

file\_stream.open(log\_path);

if (!file\_stream.is\_open()) {

std::string error\_msg = "Can't open file with path: " + log\_path;

throw new std::invalid\_argument(error\_msg);

}

log\_count = 0;

}

Logger& Logger::**getInstance**() {

if (logger\_inst == nullptr) {

logger\_inst = new Logger();

}

return \*logger\_inst;

}

void Logger::**setLogPath**(std::string new\_path) {

log\_path = new\_path;

}

Logger& Logger::operator << (LogFile& log\_info) {

if (log\_info.getStyle() & LogStyle::kNumbered)

log\_count++;

file\_stream << stylize\_log(log\_info);

file\_stream.flush();

return \*logger\_inst;

}

Logger& Logger::operator << (LogConsole& log\_info) {

if (log\_info.getStyle() & LogStyle::kNumbered)

log\_count++;

std::cout << stylize\_log(log\_info);

fflush(stdout);

return \*logger\_inst;

}

std::string Logger::**stylize\_log**(Log& log\_info) {

std::string message = "";

if (log\_info.getStyle() & LogStyle::kCapitalise) {

message = log\_info.getMessage();

for (auto it = message.begin(); it < message.end(); it++) {

\*it = toupper(\*it);

}

} else {

message = log\_info.getMessage();

}

if (log\_info.getStyle() & LogStyle::kTime) {

time\_t curr\_sec = time(nullptr);

std::string curr\_time = asctime(localtime(&curr\_sec));

curr\_time.pop\_back();

message = "(" + curr\_time + ") " + message;

}

if (log\_info.getStyle() & LogStyle::kNumbered) {

message = "[" + std::to\_string(log\_count) + "] " + message;

}

message += "\n";

return message;

}

Logger::~**Logger**() {

file\_stream.close();

}

Название файла: GameLogic.cpp

#include "GameLogic.h"

GameLogic::**GameLogic**() {

strategy\_list[kStandart] = new MoveStrategyStandart();

strategy\_list[kChase] = new MoveStrategyChase();

strategy\_list[kConfusion] = new MoveStrategyConfusion();

}

void GameLogic::**CreateRoom**(int id, int is\_new\_or\_load) {

// 0 - new; not 0 - load

if (!is\_new\_or\_load) {

QFile room\_file(QStringLiteral(":/Rooms/Data/Rooms/%1\_room.txt").arg(id));

curr\_room\_ = new Room(id, *room\_file*);

room\_file.*close*();

LogInfo(LogMode::kFile, LogStyle::kTime,

"Room created!\n|-->Width: " + std::to\_string(curr\_room\_->getWidth()) +

"\n|-->Height: " + std::to\_string(curr\_room\_->getHeight()));

QFile objects\_file(QStringLiteral(":/RoomObjects/Data/RoomObjects/%1\_objects.txt").arg(id));

LoadCellObjects(*objects\_file*);

objects\_file.*close*();

}

NotifyRoomChange();

NotifyRoomObjectsChange();

}

Room& GameLogic::**getRoom**() {

return \*curr\_room\_;

}

std::vector<PlaceableInCell\*>& GameLogic::**getRoomObjects**() {

return room\_objects\_;

}

void GameLogic::**LoadCellObjects**(QFile& obj\_file) {

if (obj\_file.exists() && obj\_file.*open*(QIODevice::ReadOnly)) {

QString curr\_line = obj\_file.readLine();

QStringList list\_param;

int num\_objects = curr\_line.toInt();

PlaceableInCell\* curr\_obj;

for (int i=0; i < num\_objects; i++) {

curr\_line = obj\_file.readLine();

list\_param = curr\_line.split(" ");

switch(list\_param[0].toInt()) {

case(1):

curr\_room\_->getRoomCell(list\_param[1].toInt(), list\_param[2].toInt()).setObject(\

*&*Player*::getInstance()*);

Player::getInstance().*Move*(Pos{list\_param[1].toInt(), list\_param[2].toInt()});

Player::getInstance().setSanity(list\_param[3].toInt());

Player::getInstance().damage\_to\_enemy\_ = list\_param[4].toInt();

room\_objects\_.push\_back(&Player::getInstance());

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"Player added: \n|-->Position: (" + std::to\_string(list\_param[1].toInt())

+ ", " + std::to\_string(list\_param[2].toInt()) + ")" +

"\n|-->Sanity: " + std::to\_string(list\_param[3].toInt()) +

"\n|-->Damage to enemies: " + std::to\_string(list\_param[4].toInt()));

break;

case(2):

switch(list\_param[1].toInt()) {

case(1):

curr\_obj = new DamageableEnemy(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()},

list\_param[5].toInt(), list\_param[6].toInt(), *strategy\_list[(*MoveStrategies*)list\_param[*7*].toInt()]*);

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"DamageableEnemy added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Image id: " + std::to\_string(list\_param[2].toInt()) +

"\n|-->Attack: " + std::to\_string(list\_param[5].toInt()) +

"\n|-->Life: " + std::to\_string(list\_param[6].toInt()));

break;

case(2):

curr\_obj = new LimitedLifeEnemy(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()},

list\_param[5].toInt(), list\_param[6].toInt(), *strategy\_list[(*MoveStrategies*)list\_param[*7*].toInt()]*);

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"LimitedLifeEnemy added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Image id: " + std::to\_string(list\_param[2].toInt()) +

"\n|-->Attack: " + std::to\_string(list\_param[5].toInt()) +

"\n|-->Life: " + std::to\_string(list\_param[6].toInt()));

break;

case(3):

curr\_obj = new ImmortalEnemy(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()},

list\_param[5].toInt(), *strategy\_list[(*MoveStrategies*)list\_param[*6*].toInt()]*);

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"ImmortalEnemy added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Image id: " + std::to\_string(list\_param[2].toInt()) +

"\n|-->Attack: " + std::to\_string(list\_param[5].toInt()) +

"\n|-->Life: Inf");

break;

}

break;

case(3):

switch(list\_param[1].toInt()) {

case(1):

curr\_obj = new SanityAffectItem(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()},

list\_param[5].toInt());

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"SanityAffectItem added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Image id: " + std::to\_string(list\_param[2].toInt()) +

"\n|-->Sanity heal: " + std::to\_string(list\_param[5].toInt()));

break;

case(2):

curr\_obj = new DamageToEnemiesItem(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()},

list\_param[5].toInt());

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"DamageToEnemiesItem added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Image id: " + std::to\_string(list\_param[2].toInt()) +

"\n|-->Damage up: " + std::to\_string(list\_param[5].toInt()));

break;

case(3):

curr\_obj = new KeyItem(list\_param[2].toInt(), Pos{list\_param[3].toInt(), list\_param[4].toInt()});

LogInfo(LogMode::kFile, LogStyle::kNumbered | LogStyle::kTime,

"KeyItem added: \n|-->Position: (" + std::to\_string(list\_param[3].toInt())

+ ", " + std::to\_string(list\_param[4].toInt()) + ")" +

"\n|-->Id: " + std::to\_string(list\_param[2].toInt()));

break;

}

break;

}

if (list\_param[0].toInt() != 1) {

room\_objects\_.push\_back(curr\_obj);

curr\_room\_->getRoomCell(list\_param[3].toInt(), list\_param[4].toInt()).setObject(*curr\_obj*);

}

}

}

}

void GameLogic::**MakeTurn**(Pos player\_pos\_change) {

Pos last\_pos, curr\_pos;

for (int i=0; i < room\_objects\_.size(); i++) {

last\_pos = room\_objects\_[i]->getPos();

if (room\_objects\_[i] == &Player::getInstance()) {

room\_objects\_[i]->*Move*(Pos{last\_pos.x + player\_pos\_change.x, last\_pos.y + player\_pos\_change.y});

} else {

room\_objects\_[i]->*Move*();

}

curr\_pos = room\_objects\_[i]->getPos();

room\_objects\_[i]->setPos(last\_pos);

if (curr\_pos.x == last\_pos.x && curr\_pos.y == last\_pos.y) {

continue;

}

if (curr\_pos.x < curr\_room\_->getWidth() && curr\_pos.x >= 0 && \

curr\_pos.y < curr\_room\_->getHeight() && curr\_pos.y >= 0) {

if (curr\_room\_->getRoomCell(curr\_pos).getObject() != nullptr) {

PlaceableInCell\* interact\_obj = curr\_room\_->getRoomCell(curr\_pos).getObject();

Interactions interaction = room\_objects\_[i]->*Interact*(*interact\_obj*);

curr\_room\_->getRoomCell(room\_objects\_[i]->getPos()).setObject(*room\_objects\_[i]*);

curr\_room\_->getRoomCell(interact\_obj->getPos()).setObject(*interact\_obj*);

if (interaction == kI\_interact\_P) {

curr\_room\_->getRoomCell(last\_pos).setObject(nullptr);

delete room\_objects\_[i];

room\_objects\_.erase(room\_objects\_.begin() + i);

NotifyObjectDestroy(i);

LogInfo(LogMode::kFile, LogStyle::kTime,

"Player interact with item at position: (" + std::to\_string(last\_pos.x) +

", " + std::to\_string(last\_pos.y) + ")");

} else if (interaction == kP\_interact\_I) {

auto interact\_obj\_pos = std::find(room\_objects\_.begin(), room\_objects\_.end(), interact\_obj);

curr\_room\_->getRoomCell(last\_pos).setObject(nullptr);

curr\_room\_->getRoomCell(room\_objects\_[i]->getPos()).setObject(*room\_objects\_[i]*);

delete interact\_obj;

room\_objects\_.erase(interact\_obj\_pos);

NotifyObjectDestroy(interact\_obj\_pos - room\_objects\_.begin());

LogInfo(LogMode::kFile, LogStyle::kTime,

"Player interact with item at position: (" + std::to\_string(room\_objects\_[i]->getPos().x) +

", " + std::to\_string(room\_objects\_[i]->getPos().y) + ")");

} else {

LogInfo(LogMode::kFile, LogStyle::kTime,

"Object interact another object, positions: (" + std::to\_string(room\_objects\_[i]->getPos().x) +

", " + std::to\_string(room\_objects\_[i]->getPos().y) + ") <--> (" +

std::to\_string(interact\_obj->getPos().x) + ", " + std::to\_string(interact\_obj->getPos().y) + ")");

}

} else {

if (room\_objects\_[i] == &Player::getInstance()) {

bool end = curr\_room\_->getRoomCell(curr\_pos).*PlayerInteract*(Player*::getInstance()*);

if (Player::getInstance().getPos() == curr\_pos) {

room\_objects\_[i]->setPos(curr\_pos);

curr\_room\_->getRoomCell(curr\_pos).setObject(*room\_objects\_[i]*);

curr\_room\_->getRoomCell(last\_pos).setObject(nullptr);

}

if (end) {

NotifyWin();

}

} else {

room\_objects\_[i]->setPos(curr\_pos);

curr\_room\_->getRoomCell(curr\_pos).setObject(*room\_objects\_[i]*);

curr\_room\_->getRoomCell(last\_pos).setObject(nullptr);

}

}

}

}

NotifyTurnComplete();

}

GameLogic::~**GameLogic**() {

auto it = strategy\_list.begin();

while (it != strategy\_list.end()) {

delete it->second;

it = strategy\_list.erase(it);

}

delete curr\_room\_;

delete &Logger::getInstance();

}

Название файла: GameView.cpp

#include "GameView.h"

#include "ui\_GameView.h"

GameView::**GameView**(GameLogic\* new\_logic, QWidget \*parent) :

QWidget(*parent*), ui(new Ui::GameView) {

ui->setupUi(this);

curr\_logic\_ = new\_logic;

curr\_logic\_->setObserver(this);

room\_scene = new QGraphicsScene(this);

ui->room\_view->setScene(*room\_scene*);

ui->room\_view->scale(0.4, 0.4);

ui->room\_view->setFocusPolicy(Qt::NoFocus);

ui->room\_view->setRenderHint(QPainter::Antialiasing);

ui->room\_view->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

ui->room\_view->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

test\_info = new QLabel(this);

test\_info->setGeometry(0, 0, 500, 700);

test\_info->setStyleSheet("QLabel {color: white;}");

}

void GameView::***resizeEvent***(QResizeEvent\* event) {

QWidget::resizeEvent(*event*);

LogInfo(LogMode::kFile, LogStyle::kTime | LogStyle::kCapitalise,

"Window resized");

}

void GameView::***keyPressEvent***(QKeyEvent\* event) {

// пока что жестко закреплены в коде, потом будут изменены

if (event->key() == Qt::Key\_Up) {

emit PlayerPressMove(Pos{0, -1});

} else if (event->key() == Qt::Key\_Down) {

emit PlayerPressMove(Pos{0, 1});

} else if (event->key() == Qt::Key\_Left) {

emit PlayerPressMove(Pos{-1, 0});

} else if (event->key() == Qt::Key\_Right) {

emit PlayerPressMove(Pos{1, 0});

}

}

void GameView::***UpdateRoomChange***() {

DeleteRoom();

Room& curr\_room = curr\_logic\_->getRoom();

int width\_room = curr\_room.getWidth();

int height\_room = curr\_room.getHeight();

QPixmap floor\_img(QStringLiteral(":/Img/Data/Img/Floor/%1\_floor.jpg").arg(curr\_room.getId()));

QPixmap exit\_img(QStringLiteral(":/Img/Data/Img/Floor/%1\_exit.jpg").arg(curr\_room.getId()));

QPixmap entrance\_img(QStringLiteral(":/Img/Data/Img/Floor/%1\_entrance.jpg").arg(curr\_room.getId()));

floor\_img = floor\_img.scaled(floor\_cell\_size\_, floor\_cell\_size\_, Qt::KeepAspectRatio);

exit\_img = exit\_img.scaled(floor\_cell\_size\_, floor\_cell\_size\_, Qt::KeepAspectRatio);

entrance\_img = entrance\_img.scaled(floor\_cell\_size\_, floor\_cell\_size\_, Qt::KeepAspectRatio);

for (int x = 0; x < width\_room; x++) {

for (int y = 0; y < height\_room; y++) {

QPixmap chosen\_img;

if (typeid(curr\_room.getRoomCell(x, y)) == typeid(ExitCell)) {

chosen\_img = exit\_img;

} else if (typeid(curr\_room.getRoomCell(x, y)) == typeid(EntranceCell)) {

chosen\_img = entrance\_img;

} else {

chosen\_img = floor\_img;

}

QGraphicsPixmapItem\* floor\_item = room\_scene->addPixmap(chosen\_img);

floor\_item->setPos(x\*floor\_cell\_size\_, y\*floor\_cell\_size\_ + top\_padding\_scene\_);

}

}

}

void GameView::***UpdateRoomObjectsChange***() {

ClearObjects();

auto room\_obj = curr\_logic\_->getRoomObjects();

for (int i=0; i < room\_obj.size(); i++) {

draw\_objects\_.push\_back(new DrawableObject(*room\_scene*, room\_obj[i]->getId(), room\_obj[i]->getPos(), floor\_cell\_size\_, top\_padding\_scene\_));

}

CenterViewOnPlayerPos();

}

void GameView::***UpdateObjectDestroy***(int i) {

room\_scene->removeItem(*draw\_objects\_[i]*);

delete draw\_objects\_[i];

draw\_objects\_.erase(draw\_objects\_.begin() + i);

}

void GameView::***UpdateTurnComplete***() {

Room& curr\_room = curr\_logic\_->getRoom();

int width\_room = curr\_room.getWidth();

int height\_room = curr\_room.getHeight();

for (int x = 0; x < width\_room; x++) {

for (int y = 0; y < height\_room; y++) {

if (curr\_room.getRoomCell(x, y).getObject()) {

int i = std::find\_if(curr\_logic\_->getRoomObjects().begin(), curr\_logic\_->getRoomObjects().end(),

[x, y](PlaceableInCell\* curr\_obj) {

return curr\_obj->getPos().x == x && curr\_obj->getPos().y == y;

}) - curr\_logic\_->getRoomObjects().begin();

QPropertyAnimation\* anim = new QPropertyAnimation(*draw\_objects\_[i]*, "getScenePos");

anim->setDuration(200);

anim->setEndValue(draw\_objects\_[i]->RoomPosToScenePos(curr\_logic\_->getRoomObjects()[i]->getPos()));

anim->start(QAbstractAnimation::DeleteWhenStopped);

}

}

}

CenterViewOnPlayerPos();

TestShowPlayerConditions();

}

void GameView::***UpdateWin***() {

QLabel\* win = new QLabel("You win!");

win->show();

}

void GameView::***UpdateDefeat***() {

QLabel\* defeat = new QLabel("You lose... your sanity ;)");

defeat->show();

}

void GameView::**TestShowPlayerConditions**() {

QString info = QStringLiteral("Player: "\

"Sanity: %1\nDamage\_to\_enemies: %2\nNum\_key\_items: %3\n\n").arg(Player::getInstance().getSanity()

).arg(Player::getInstance().damage\_to\_enemy\_).arg(Player::getInstance().getNumItems());

test\_info->setText(info);

}

void GameView::**CenterViewOnPlayerPos**() {

Pos player\_pos = Player::getInstance().getPos();

ui->room\_view->centerOn(floor\_cell\_size\_\* player\_pos.x, floor\_cell\_size\_\* player\_pos.y + floor\_cell\_size\_);

}

void GameView::**DeleteRoom**() {

ClearObjects();

room\_scene->clear();

}

void GameView::**ClearObjects**() {

for (int i=0; i < draw\_objects\_.size(); i++) {

room\_scene->removeItem(*draw\_objects\_[i]*);

delete draw\_objects\_[i];

}

draw\_objects\_.clear();

}

GameView::~***GameView***() {

DeleteRoom();

delete room\_scene;

delete ui;

}

# Приложение б ТЕСТИРОВАНИЕ

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Выполняемые операции | Результат | Комментарии |
|  | Логирование в консоль в начале игры [логируются *GameLogic* и *GameView*]. | См. рис. 1 | Верно |
|  | Логирование в консоль, несколько ходов прошло [логируются *GameLogic* и *GameView*]. | См. рис. 2 | Верно |
|  | Логирование в файл (открыт после завершения игры) [логируются *GameLogic* и *GameView*]. | См. рис. 3 | Верно |
|  | Логирование в файл и в консоль всей информации [логируются *GameLogic* и *GameView*]. | См. рис. 4 | Верно |
|  | Смена формата логов для события изменения экрана приложения + прекращение отслеживания GameLogic изменений [логируется *GameView*]. | См. рис. 5 | Верно |
|  | Смена всех форматов для GameLogic + прекращение отслеживания *GameView* изменений [логируется GameLogic]. | См. рис. 6 | Верно |

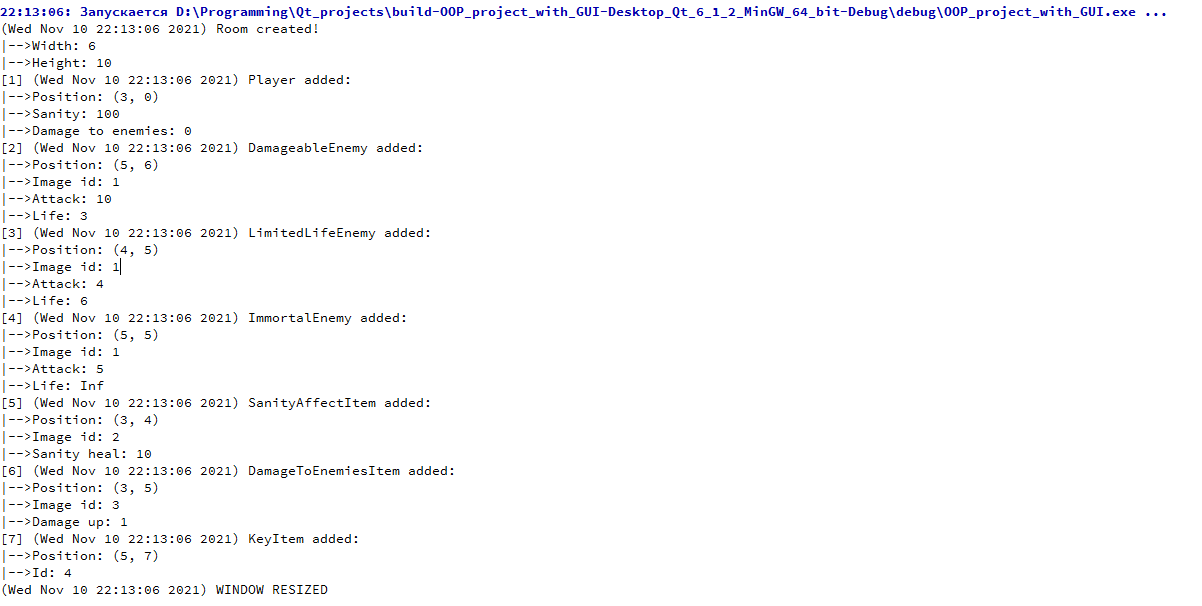


Рисунок 1 - Лог, консоль, старт игры

|  |  |
| --- | --- |
|  |  |

Рисунок 2 - Лог, консоль, продолжение игры

|  |  |
| --- | --- |
|  |  |

Рисунок 3 - Лог, файл, продолжение игры

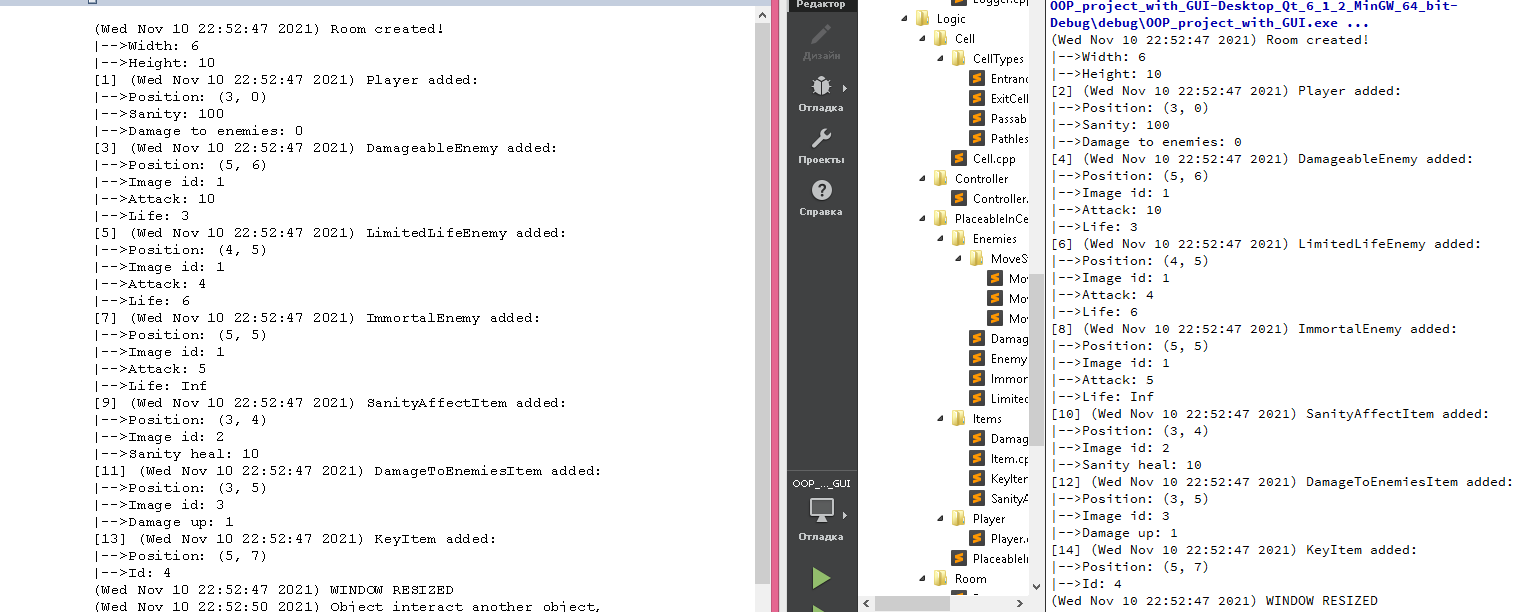


Рисунок 4 - Часть лога, файл и консоль

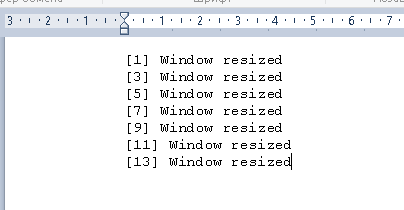


Рисунок 5 - Лог, отслеживание только GameView, файл

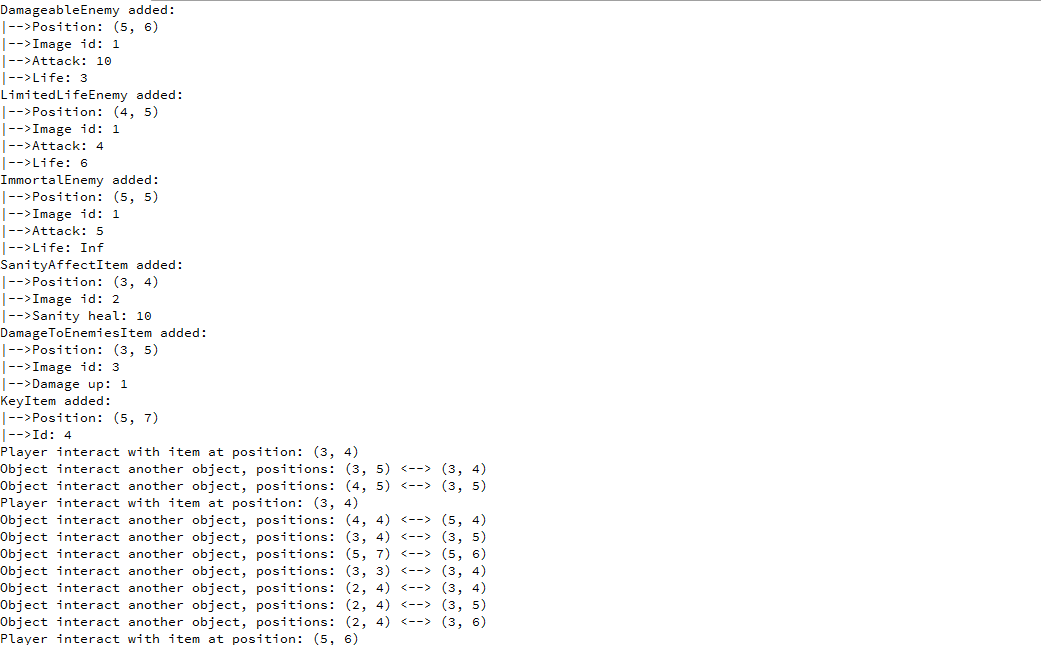


Рисунок 6 - Лог, отслеживание только GameLogic, консоль

# Приложение В uml-ДИАГРАММА

UML-диаграмма классов представлена на рис. 5:

