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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: Шаблонные классы, управление

Студент гр. 0303 _____ Парамонов В.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Добавить в проект шаблонные классы правил игры, на которых базируются ограничения и условия продолжения игрока, и класс игры, который должен управлять связью бизнес-логики и команд управления.

Задание.

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и.т.д.). Затем определить класс игры, который параметризуется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

- Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
- Создан шаблонный класс игры, который параметризуется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и.т.д.

Основные теоретические положения.

Потенциальные паттерны проектирования, которые можно использовать:

- Компоновщик (*Composite*) - выстраивание иерархии правил.
- Фасад (*Facade*) - предоставления единого интерфейса игры для команд управления.
- Цепочка обязанностей (*Chain of Responsibility*) - обработка поступающих команд управления.
- Состояние (*State*) - отслеживание состояние хода / передача хода от игрока к врагам.

Посредник (*Mediator*) - организация взаимодействия элементов бизнес-логики.

Выполнение работы.

1) Шаблонные классы правил:

- Класс *Rule* – базовый абстрактный класс для правил, содержащий две основные виртуальные функции, которые должны быть в каждом правиле: *CheckRuleCondition*(int check_curr_room, PlaceableInCell* check_obj), которая нужна для проверки выполнения условия правила, вызывается при каждом уничтожении объекта в текущей (все правила на этом основаны), *CheckRule*(int check_curr_room, int check_next_room), которая должна возвращать значение поля rule_done_, олицетворяющее выполнено правило игры или еще нет.
- Класс TemplateRule – наследник класса *Rule*, в который уже добавлен шаблон из двух значений <int curr_room_id, int next_room_id>, первое значение – комната, в которой действует правило, второе значение – комната, в которую можно попасть из данной только после выполнения правила. Этот класс также реализует функцию *CheckRule*(int check_curr_room, int check_next_room), которая проверяет та ли это комната и запрашивается ли переход в нужную комнату, если да, то возвращает rule_done_, иначе возвращает *true* (так как для других переходов и комнат данное правило не должно влиять).
- Теперь рассмотрим сразу 3 класса – KeyItemRule <int curr_room_id, int next_room_id, int item_id>, DestroyObjectRule <int curr_room_id, int next_room_id, int obj_id>, DestroyedObjectNumRule <int curr_room_id, int next_room_id, int num_obj>. Все они наследуются от TemplateRule и реализуют функцию CheckRuleCondition, работающую по-разному для каждого класса: для KeyItemRule при уничтожении объекта идет проверка на то, добавился ли этот объект в инвентарь игрока, если да, то идет проверка на соответствие индекса объекта с индексом item_id (переданным в

шаблон), для DestroyObjectRule при уничтожении объекта идет проверка на соответствие его индекса и obj_id, для DestroyedObjectNumRule идет увеличение поля destroyed_obj_num_, если значение этого поля больше или равно num_obj, то условие выполнено.

Все 3 класса еще перед проверкой основного условия проверяют в нужной ли комнате происходят события (curr_room_id == check_curr_room?) и выполнено ли это правило или еще нет. Также для удобства отслеживания выполнение правил все они логируются с помощью логера из прошлой лабораторной.

- Класс RulesList служит в качестве контейнера для хранения всех правил игры, имеет основной метод AddRule(Rule* new_rule), для добавления правил, а также 2 метода для вызова практически одноименных методов у всех хранимых правил: CheckRules(int check_curr_room, int check_next_room), CheckRulesCondition(int check_curr_room, PlaceableInCell* check_obj). CheckRules возвращает логическое значение, которое означает возможен ли переход из комнаты в комнату (выполнены ли все правила).

2) Класс всей игры:

- Класс Game <RulesList& rules> – олицетворение всей игры, хранит в своих полях основные структурные элементы проекта: бизнес-логику (GameLogic logic_), графический интерфейс (GameView view_), устройство управления (Controller controller_), а также состояния посещенных комнат (std::map<int, RoomState> load_rooms_), для реализации перехода между ними. Метод **Start**() – запускает игру; RoomChanging(int id) – обрабатывает изменение комнаты игроком, если до этого он уже посещал комнату, в которую собирается идти, то вызывается метод логики для загрузки комнаты, если нет, то она создается из файла логикой; RuleUpdate(PlaceableInCell* destroyed_obj) и RuleCheck(int

next_room_id) вызываются по сигналу логики об уничтожении объекта или инициации перехода в другую комнату, работают с переданным в шаблон RulesList и исользуют его методы CheckRules и CheckRulesCondition соответственно; PlayerChangePos(Pos player_pos_change) реагирует на сигнал контроллера о нажатии на клавиатуре клавиши перемещения игрока.

- Game наследуется от RoomChangeObserver и ControllerObserver, которые используют паттерн наблюдатель для получения сигналов от GameLogic и Controller, необходимы они в основном для сокрытия от логики и контроллера всех возможностей класс Game, чтобы они могли вызывать только созданные для них методы сообщения об изменении состояния.

3) Изменения контроллера:
   - Класс Controller, отвечающий за управление сигналами, поступающими с клавиатуры, утратил возможность напрямую передавать команды GameLogic, теперь он передает изменение позиции игрока в соответствии с нажатыми клавишами только наблюдателю за ним – ControllerObserver.
4) Остальные изменения, не имеющие отношения к лабораторной:
   - Класс GameLogic получил изменения некоторых методов, для реализации возможности перехода между комнатами игроком.
   - Добавлен новый наследник PlaceableInCell – класс Furniture, описывающий статичные объекты, которые не взаимодействуют с другими объектами, а просто служат декором (однако могут занимать несколько клеток). Все взаимодействия с полем и графическим интерфейсом были прописаны.

Разработанный программный код см. в приложении А.

Тестирование см. в приложении Б.

UML-диаграмму см. в приложении В.

Выводы.

Были реализованы 3 шаблонных правила игры, а также сам класс игры, параметризуемый конкретными правилами. Все проверки на выполнение условий работают, выполнение правил засчитывается.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <QApplication>

#include "Game/Game.h"
#include "Logic/Rule/DestroyedObjectNumRule.h"
#include "Logic/Rule/DestroyObjectRule.h"
#include "Logic/Rule/KeyItemRule.h"
#include "Logic/Rule/RulesList.h"

static RulesList all_rules = RulesList();

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Game<all_rules> game;
    all_rules.AddRule(new DestroyedObjectNumRule<2, 1, 2>());
    all_rules.AddRule(new KeyItemRule<1, 2, 4>());
    all_rules.AddRule(new KeyItemRule<1, 6, 4>());
    all_rules.AddRule(new DestroyObjectRule<3, 1, 1>());
    game.Start();
    return a.exec();
}
```

Название файла: Rule.h

```
#ifndef RULE_H
#define RULE_H

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

class Rule: public Loggable {
protected:
    bool rule_done_ = false;
public:
    virtual bool CheckRule(int check_curr_room, int check_next_room) = 0;
    virtual void CheckRuleCondition(int check_curr_room, PlaceableInCell*
check_obj) = 0;
};

#endif // RULE_H
```

Название файла: TemplateRule.h

```
#ifndef TEMPLATERULE_H
#define TEMPLATERULE_H

#include "Rule.h"

template <int curr_room_id, int next_room_id>
class TemplateRule: public Rule {
public:
    TemplateRule() {};
```

```
    bool CheckRule(int check_curr_room, int check_next_room) {
        if (curr_room_id == check_curr_room && next_room_id ==
check_next_room) {
            return rule_done_;
        }
        return true;
    }
};

#endif // TEMPLATERULE_H
```

Название файла: KeyItemRule.h

```
#ifndef KEYITEMRULE_H
#define KEYITEMRULE_H

#include "TemplateRule.h"
#include "../PlaceableInCell/Player/Player.h"

template <int curr_room_id, int next_room_id, int item_id>
class KeyItemRule: public TemplateRule<curr_room_id, next_room_id> {
public:
    KeyItemRule() {}

    void CheckRuleCondition(int check_curr_room, PlaceableInCell*
check_obj) {
        if (!TemplateRule<curr_room_id, next_room_id>::rule_done_) {
            if (Player::getInstance().HasItem(item_id)) {
                TemplateRule<curr_room_id, next_room_id>::rule_done_ =
true;

                TemplateRule<curr_room_id,
next_room_id>::LogInfo(LogMode::kConsole, LogStyle::kTime,
                "(KeyItemRule) !is completed! passage between " +
std::to_string(curr_room_id) + " and " +
                 std::to_string(next_room_id) + " rooms now opened");
            }
        }
    }
};

#endif // KEYITEMRULE_H
```

Название файла: DestroyObjectRule.h

```
#ifndef DESTROYOBJECTRULE_H
#define DESTROYOBJECTRULE_H

#include "TemplateRule.h"

template <int curr_room_id, int next_room_id, int obj_id>
class DestroyObjectRule: public TemplateRule<curr_room_id, next_room_id>
{
public:
    DestroyObjectRule() {}
```

```
    void CheckRuleCondition(int check_curr_room, PlaceableInCell*
check_obj) {
        if (!TemplateRule<curr_room_id, next_room_id>::rule_done_ &&
check_curr_room == curr_room_id) {
            if (check_obj->getId() == obj_id) {
                TemplateRule<curr_room_id, next_room_id>::rule_done_ =
true;
            }

            TemplateRule<curr_room_id,
next_room_id>::LogInfo(LogMode::kConsole, LogStyle::kTime,
            "(DestroyObjectRule) !is completed! passage between " +
std::to_string(curr_room_id) + " and " +
             std::to_string(next_room_id) + " rooms now opened");
        }
    }
};

#endif // DESTROYOBJECTRULE_H
```

Название файла: DestroyedObjectNumRule.h

```
#ifndef DESTROYEDOBJECTNUMRULE_H
#define DESTROYEDOBJECTNUMRULE_H

#include "TemplateRule.h"

template <int curr_room_id, int next_room_id, int num_obj>
class DestroyedObjectNumRule: public TemplateRule<curr_room_id,
next_room_id> {
private:
    int destroyed_obj_num_;
public:
    DestroyedObjectNumRule() {
        destroyed_obj_num_ = 0;
    }

    void CheckRuleCondition(int check_curr_room, PlaceableInCell*
check_obj) {
        if (!TemplateRule<curr_room_id, next_room_id>::rule_done_ &&
check_curr_room == curr_room_id) {
            destroyed_obj_num_ += 1;

            TemplateRule<curr_room_id,
next_room_id>::LogInfo(LogMode::kConsole, LogStyle::kTime,
            "(DestroyedObjectNumRule) destroyed obj num now:" +
std::to_string(destroyed_obj_num_) + ", need: " +
             std::to_string(num_obj));

            if (destroyed_obj_num_ >= num_obj) {
                TemplateRule<curr_room_id, next_room_id>::rule_done_ =
true;

                TemplateRule<curr_room_id,
next_room_id>::LogInfo(LogMode::kConsole, LogStyle::kTime,
                "(DestroyedObjectNumRule) !is completed! passage between
" + std::to_string(curr_room_id) + " and " +
```

```
                std::to_string(next_room_id) + " rooms now opened");
            }
        }
    }
};

#endif // DESTROYEDOBJECTNUMRULE_H
```

Название файла: RulesList.h

```
#ifndef RULESLIST_H
#define RULESLIST_H

#include <vector>

#include "Rule.h"

class RulesList {
private:
    std::vector<Rule*> rules_;
public:
    RulesList();

    void AddRule(Rule* new_rule);

    bool CheckRules(int check_curr_room, int check_next_room);
    void CheckRulesCondition(int check_curr_room, PlaceableInCell*
check_obj);

    ~RulesList();
};

#endif // RULESLIST_H
```

Название файла: RulesList.cpp

```
#include "RulesList.h"

RulesList::RulesList() {}

void RulesList::AddRule(Rule* new_rule) {
    rules_.push_back(new_rule);
}

bool RulesList::CheckRules(int check_curr_room, int check_next_room) {
    for (Rule* p: rules_) {
        if (!p->CheckRule(check_curr_room, check_next_room)) {
            return false;
        }
    }
    return true;
}

void RulesList::CheckRulesCondition(int check_curr_room, PlaceableInCell*
check_obj) {
    for (Rule* p: rules_)
```

```
        p->CheckRuleCondition(check_curr_room, check_obj);
}

RulesList::~RulesList() {
    for (Rule* p: rules_)
        delete p;
}
```

Название файла: ControllerObserver.h

```
#ifndef CONTROLLEROBSERVER_H
#define CONTROLLEROBSERVER_H

#include "../../Logic/Room/Pos.h"

class ControllerObserver {
public:
    virtual void PlayerChangePos(Pos player_pos_change) = 0;
};

#endif // CONTROLLEROBSERVER_H
```

Название файла: Game.h

```
#ifndef GAME_H
#define GAME_H

#include <map>

#include "../Logic/GameLogic.h"
#include "../Logic/Controller/Controller.h"
#include "../Logic/Rule/RulesList.h"
#include "../Gui/GameView.h"
#include "RoomChangeNotify/RoomChangeObserver.h"
#include "ControllerObserver/ControllerObserver.h"
#include "RoomState.h"

template <RulesList& rules>
class Game: public RoomChangeObserver, public ControllerObserver {
    GameLogic logic_;
    GameView view_;
    Controller controller_;

    std::map<int, RoomState> load_rooms_;
public:
    Game(): view_(&logic_), controller_(this, &view_) {
        logic_.setRoomChangeObserver(this);
    }

    void Start() {
        logic_.CreateRoom();
        view_.show();
    }

    void RoomChanging(int id) {
```

```
        load_rooms_[logic_.getRoom().getId()] = Room-
State{&logic_.getRoom(), log-
ic_.getRoomMoveObjects(),logic_.getRoomStaticObjects() };
        if (load_rooms_.find(id) != load_rooms_.end()) {
            RoomState load_room = load_rooms_.at(id);
            logic_.LoadRoom(load_room.room, load_room.room_move_obj,
load_room.room_static_obj, logic_.getRoom().getId());
        } else {
            logic_.CreateRoom(id, logic_.getRoom().getId());
        }
    }

    void RuleUpdate(PlaceableInCell* destroyed_obj) {
        rules.CheckRulesCondition(logic_.getRoom().getId(), de-
stroyed_obj);
    }

    bool RuleCheck(int next_room_id) {
        rules.CheckRules(logic_.getRoom().getId(), next_room_id);
    }

    void PlayerChangePos(Pos player_pos_change) {
        logic_.MakeTurn(player_pos_change);
    }
};

#endif // GAME_H
```

Название файла: Controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include <QObject>

#include "../../Game/ControllerObserver/ControllerObserver.h"
#include "../../Gui/GameView.h"

class Controller: public QObject {
private:
    Q_OBJECT
    ControllerObserver* obs_;
public:
    Controller(ControllerObserver* logic_, GameView* view);
public slots:
    void MoveArrowPushed(Pos player_pos_change);
};

#endif // CONTROLLER_H
```

Название файла: Controller.cpp

```
#include "Controller.h"

Controller::Controller(ControllerObserver* new_obs_, GameView* view):
obs_(new_obs_) {
```

```
    QObject::connect(view, SIGNAL(PlayerPressMove(Pos)), this,
SLOT(MoveArrowPushed(Pos)));
}

void Controller::MoveArrowPushed(Pos player_pos_change) {
    obs_->PlayerChangePos(player_pos_change);
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Выполняемые операции	Результат	Комментарии
1.	Учет правила на уничтожения двух объектов в комнате 2.	См. рис. 1	Верно
2.	Учет правила на уничтожение объекта с определенным *id* в комнате 3.	См. рис. 2	Верно
3.	Учет правила на подбор ключевого предмета в комнате 1.	См. рис. 3	Верно

```
[3] Room changed to - 2
Room created!
|-->Width: 5
|-->Height: 5
Player added:
|-->Position: (0, 1)
|-->Sanity: 100
|-->Damage to enemies: 1
DamageableEnemy added:
|-->Position: (3, 4)
|-->Image id: 1
|-->Attack: 3
|-->Life: 4
LimitedLifeEnemy added:
|-->Position: (3, 3)
|-->Image id: 1
|-->Attack: 20
|-->Life: 25
Object interact another object, positions: (1, 1) <--> (2, 0)
Object interact another object, positions: (2, 0) <--> (1, 1)
Object interact another object, positions: (1, 1) <--> (2, 0)
(Thu Nov 25 02:25:54 2021) (DestroyedObjectNumRule) destroyed obj num now:1, need: 2
Object destroyed at pos: (0, 0)
Object interact another object, positions: (1, 0) <--> (2, 0)
Object interact another object, positions: (1, 0) <--> (2, 0)
Object interact another object, positions: (1, 0) <--> (2, 0)
(Thu Nov 25 02:26:06 2021) (DestroyedObjectNumRule) destroyed obj num now:2, need: 2
(Thu Nov 25 02:26:06 2021) (DestroyedObjectNumRule) !is completed! passage between 2 and 1 rooms now opened
Object destroyed at pos: (2, 0)
```

Рисунок 1 - Консоль после выполнения

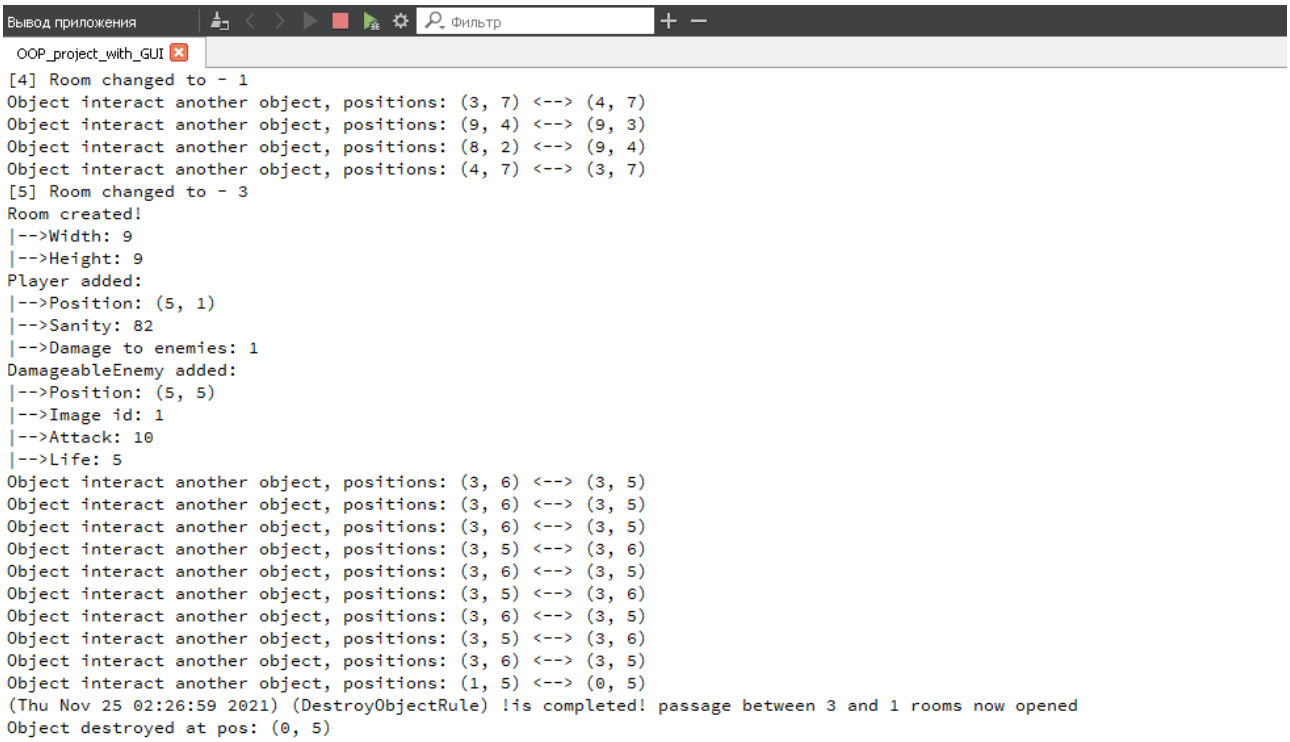

Рисунок 2 - Консоль после выполнения

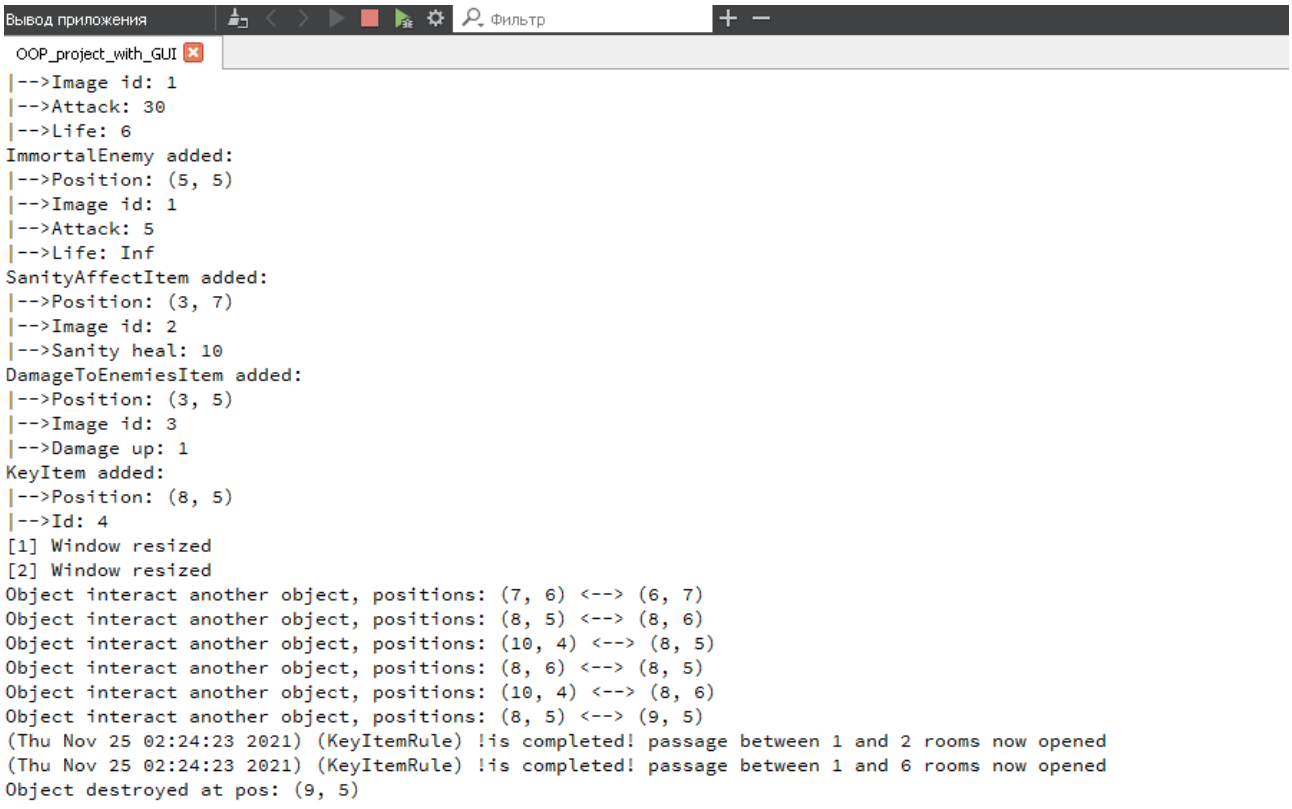

Рисунок 3 - Консоль после выполнения

ПРИЛОЖЕНИЕ В
UML-ДИАГРАММА

UML-диаграмма классов представлена на рис. 4:

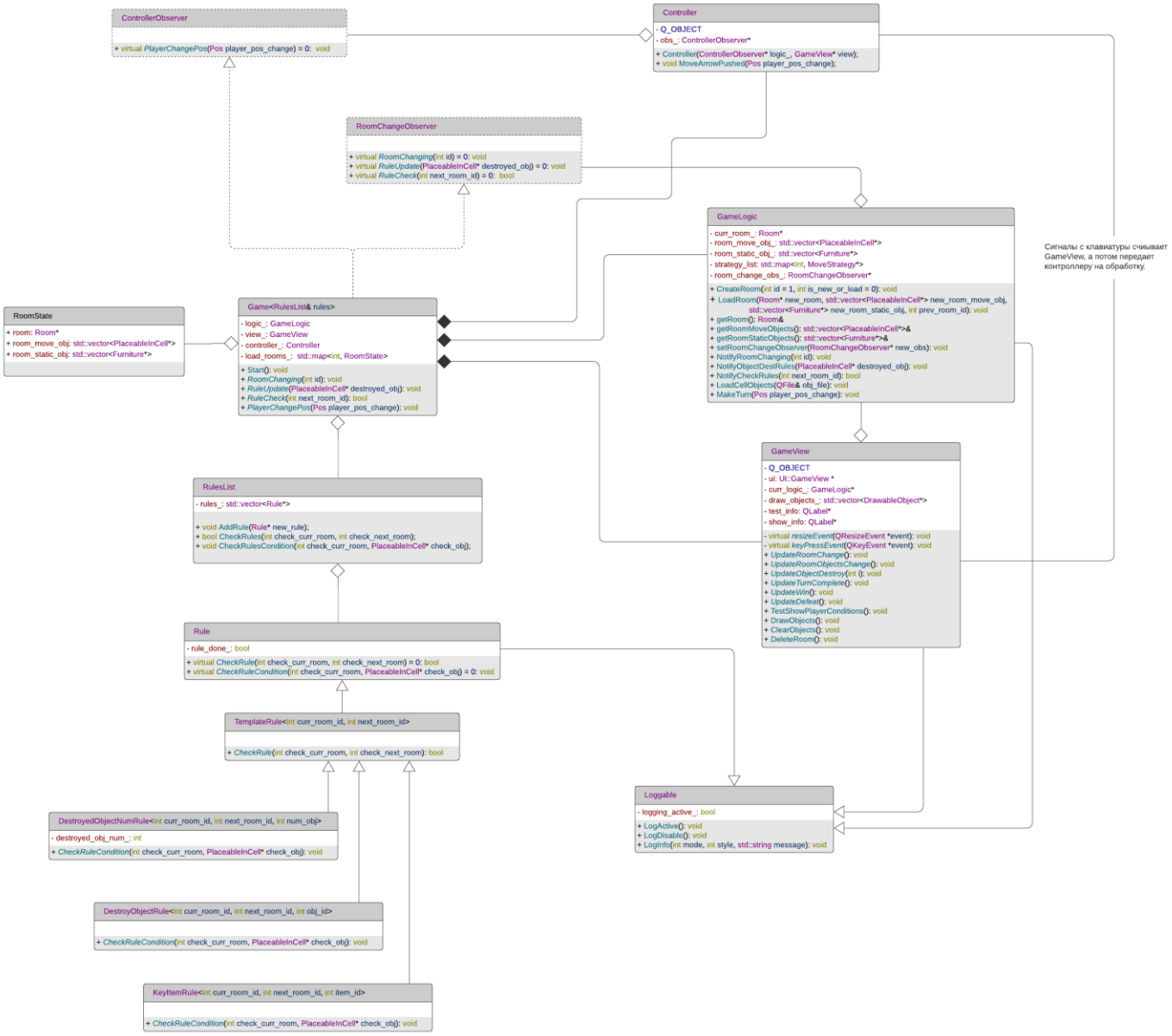

Рисунок 4 - UML-диаграмма