**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 иерарихии 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).

1. 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 иницации 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.

1. 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.

1. 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()] = RoomState{&logic\_.getRoom(), logic\_.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(), *destroyed\_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 |
|  | Учет правила на уничтожения двух объектов в комнате 2. | См. рис. 1 | Верно |
|  | Учет правила на уничтожение объекта с определенным *id* в комнате 3. | См. рис. 2 | Верно |
|  | Учет правила на подбор ключевого предмета в комнате 1. | См. рис. 3 | Верно |

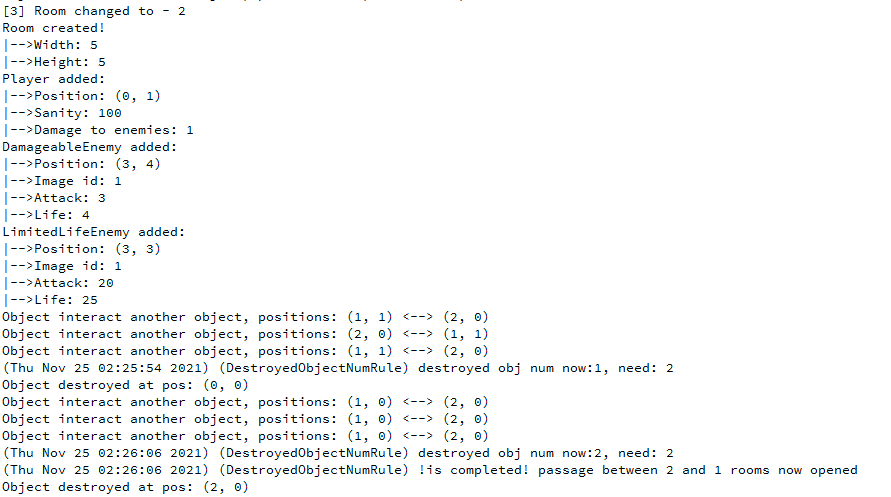


Рисунок 1 - Консоль после выполнения

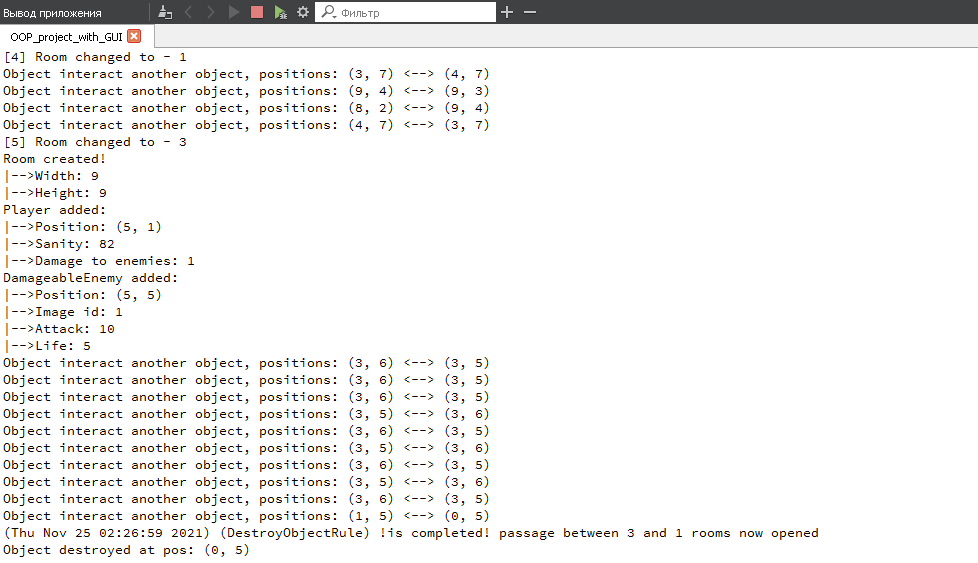


Рисунок 2 - Консоль после выполнения

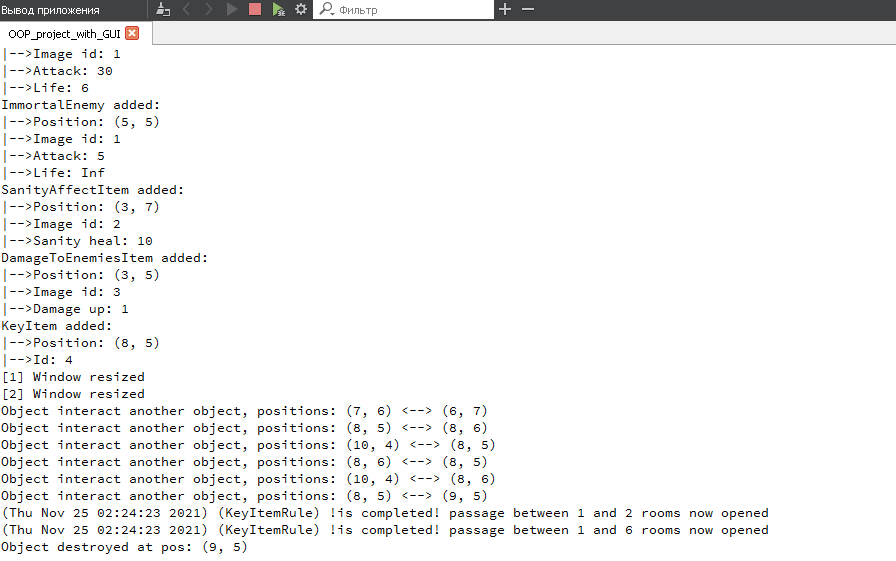


Рисунок 3 - Консоль после выполнения

# Приложение В uml-ДИАГРАММА

UML-диаграмма классов представлена на рис. 4:

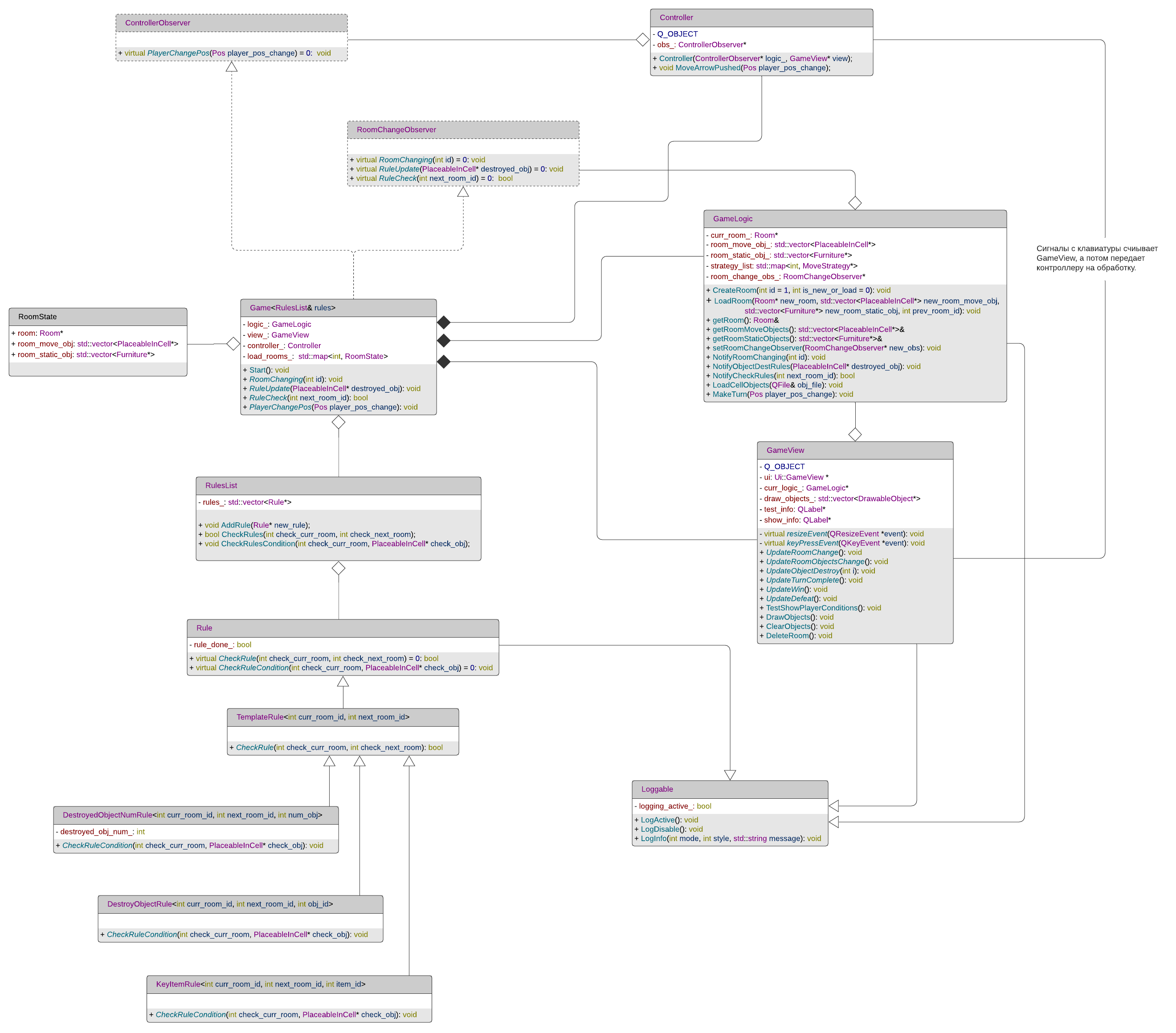


Рисунок 4 - UML-диаграмма