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: Шаблонные классы

Студент гр. 8303 _____ Хохлов Г.О.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Разработка и реализация набора классов правил игры. Основные требования:

- Правила игры должны определять начальное состояние игры
- Правила игры должны определять условия выигрыша игроков
- Правила игры должны определять очередность ходов игрока
- Должна быть возможность начать новую игру

Ход выполнения работы.

1. Была добавлена команда «new game»
2. Был изменен класс GameFacade так, что бы он принимал в качестве шаблона правила игры и количество игроков.
3. Был реализован набор правил, наследующихся от GameRule. Они определяют начальное состояние игры(ширину и высоту поля), условия выигрыша игроков и определяют очередность игроков.
4. Был реализован набор классов, наследующихся от PlayerState, которые определяют какой игрок будет ходить следующим.
5. Для класса GameFacade конструктор был перенесен в private и добавлена функция instance. Таким образом выполняется соответствие паттерну «Синглтон»

Вывод.

В ходе выполнения лабораторной работы были изучены шаблонные классы путем изменения класса GameFacade под требования лабораторной работы.

.

Приложение А. Исходный код программы

1. GameFacade.h

```
#include <sstream>
#include "GameInfo.h"
#include "UI/MainCommandInterpreter.h"

template<typename Rule, int playersCount>
class GameFacade: public GameInfo {

private:

    MainCommandInterpreter interpreter;
    GameFacade(int fieldWidth, int fieldHeight): GameInfo(playersCount,
fieldWidth, fieldWidth, new Rule){}
    Rule rule;

public:

    static GameFacade& instance(){
        Rule rule;
        static GameFacade singleInstance(rule.fieldWidth, rule.fieldHeight);
        return singleInstance;
    }

    void nextTurn(){

        std::string commandString;
        std::getline(std::cin,  commandString);

        CommandPtr command = interpreter.handle(commandString);
        command->execute(*this);
        history.push_back(command->getMemento());

        nextUser();

    }

    friend std::ostream &operator<<(std::ostream &stream, const GameFacade
&game){

        stream << "Now player: " << game.nowPlayerIndex << std::endl;
        stream << game.gameField << std::endl;
        return stream;

    }

    bool isOver(){

        return rule.isOver(*this);

    }

};
```

2. GameRule.h

```
class GameInfo;

class GameRule {

public:
    int fieldWidth;
    int fieldHeight;
    virtual bool isOver(GameInfo &gameInfo)=0;
    virtual int nextUser(GameInfo &gameInfo)=0;

    GameRule(int fieldWidth, int fieldHeight):
    fieldWidth(fieldWidth),
    fieldHeight(fieldHeight){}

};
```

3. PlayerState.h

```
class PlayerState {

public:

    virtual int getNextPlayerDelta()=0;
    virtual PlayerState* getNextPlayerState()=0;

};

typedef std::unique_ptr<PlayerState> PlayerStatePtr;

class SecondPlayer: public PlayerState {

    int getNextPlayerDelta() override{
        return 2;
    }

    PlayerState* getNextPlayerState() override{
        return nullptr;
    }

};

class ThirdPlayer: public PlayerState {

    int getNextPlayerDelta() override{
```

```
        return -1;
    }

    PlayerState* getNextPlayerState()
        override{ return new
        SecondPlayer;
    }

};

class FirstPlayer: public

PlayerState { public:

    int getNextPlayerDelta()
        override{ return 2;
    }

    PlayerState* getNextPlayerState()

        override{ return new ThirdPlayer;

    }

};
```