МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Объектно-ориентированное программирование»

Тема: управление, разделение на уровни абстракции

Студент гр. 0383: Бояркин Н.А.

Преподаватель: Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Разработать организацию управление игрой с разделением на уровни абстракции.

Задание.

Необходимо организовать управление игрой (номинально через CLI). При управлении игрой с клавиатуры должна считываться нажатая клавиша, после чего происходит перемещение игрок или его взаимодействия с другими элементами поля.

Требования:

- Реализовать управление игрой. Считывание нажатий клавиш не должно происходить в классе игры, а должно происходить в отдельном наборе классов.
- Клавиши управления не должны жестко определяться в коде. Например, это можно определить в отдельном классе.
- Классы управления игрой не должны напрямую взаимодействовать с элементами игры (поле, клетки, элементы на клетках)
- Игру можно запустить и пройти.

  Потенциальные паттерны проектирования, которые можно использовать:

- Команда (Command) - передача команд с информацией через единый интерфейс. помещение команд в очередь

- Посредник (Mediator) - организация взаимодействия различных модулей

Выполнение работы:

1) Создан общий интерфейс команд *class Command*, в котором есть только чистые виртуальные функции, а их реализация перегружена в классах *CommandUp, CommandDown, CommandLeft, CommandRight*, которые наследуются от класса *Command*. Также были созданы перечисление *Commands* для того, чтобы типы переменных *Commands* хранились в классах команд.

2) Соответственно, в классах команд перегружены функции *char execute(),* которое возвращает направление для функции moveEntity, конструктор *Command<dir>(Commands command_user)*, который изменяет поле *Command* и метод *Commands get_instruction() const*, который получается поле *Command.*

3) Разработан класс “Отправитель”, который хранит поле класса *Console*, а также указатели на класс *Command*. Класс *Invoker* работает с командами только через их общий интерфейс, то есть *Command*.

4) Разработан общий интерфейс устройств - класс *class Devices*. Он создан для того, чтобы у нас была возможность реализовать управление не только клавиатурой, но и мышкой(например).

5) Следовательно, был создан класс *Console*, который наследуется от *Devices*, в котором определяются клавиши управления игрой. В нём

создан контейнер map, который в качестве ключа принимает перечисления типа *Command*, а значения принимает типа *char*.

6) Таким образом, в функции *char run()* класса *Invoker* происходит считывание клавиш, далее программа берёт у устройства команду (т.е. принимает тип char, возвращает тип *Commands*) и сравнивает с направлением указателем на экземпляр класса *Command<dir>* и в случае, если они равны выполняется метод *execute()* у класса Command<dir>.

   Разработанный программный код см. в приложении А.

   Диаграмму с зависимостями классов см. в приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	Результат	Комментарии
1.	Есть возможность ходить налево, направо, вверх, вниз.		Верно
2.	Игру можно запустить и пройти.		Верно
3.	Герой не имеет возможности ходить с здоровьем < 0.		Верно
4.	При смерти героя, игра будет окончена.		Верно

Выводы.

В ходе работы были изучена разработка организации управлении игрой с разделением на уровни абстракции..

Приложение А.

Исходный код программы.

Название файла: Command.h

```
#pragma once
enum Commands{
    Up,
    Down,
    Left,
    Right
};

class Command{
public:
    virtual char execute() = 0;
    virtual void change(Commands instruction) = 0;
     [[nodiscard]] virtual Commands get_instruction() const =
0;
    virtual ~Command(){

    }
};
```

Название файла: CommandUp.h

```
#pragma once
#include "Command.h"

class CommandUp: public Command{
private:
    Commands command;
public:
    CommandUp();
    CommandUp(Commands command_user);
    [[nodiscard]] Commands get_instruction() const override;
    void change(Commands instruction) override;
    char execute() override;
};
```

Название файла: CommandUp.cpp

```
#include "CommandUp.h"

CommandUp::CommandUp() {
    this->command = Commands::Up;
}

CommandUp::CommandUp(Commands command_user) {
    this->command = command_user;
```

```
}

Commands CommandUp::get_instruction() const {
    return this->command;
}

char CommandUp::execute() {
    return 'W';
}

void CommandUp::change(Commands instruction) {
    this->command = instruction;
}
```

Название файла: CommandDown.h

```
#pragma once
#include "Command.h"

class CommandDown: public Command{
private:
    Commands command;
public:
    CommandDown();
    CommandDown(Commands command_user);
    [[nodiscard]] Commands get_instruction() const override;
    void change(Commands instruction) override;
    char execute() override;
};
```

Название файла: CommandDown.cpp

```
#include "CommandDown.h"

CommandDown::CommandDown() {
    this->command = Commands::Down;
}

CommandDown::CommandDown(Commands command_user) {
    this->command = command_user;
}

Commands CommandDown::get_instruction() const {
    return this->command;
}

char CommandDown::execute() {
    return 'S';
}

void CommandDown::change(Commands instruction) {
```

```
    this->command = instruction;
}
```

Название файла: CommandLeft.h

```
#pragma once
#include "Command.h"

class CommandLeft: public Command{
private:
    Commands command;
public:
    CommandLeft();
    CommandLeft(Commands command_user);
    [[nodiscard]] Commands get_instruction() const override;
    void change(Commands instruction) override;
    char execute() override;
};
```

Название файла: CommandLeft.cpp

```
#include "CommandLeft.h"

CommandLeft::CommandLeft() {
    this->command = Commands::Left;
}

CommandLeft::CommandLeft(Commands command_user) {
    this->command = command_user;
}

Commands CommandLeft::get_instruction() const {
    return this->command;
}

char CommandLeft::execute() {
    return 'A';
}

void CommandLeft::change(Commands instruction) {
    this->command = instruction;
}
```

Название файла: CommandRight.h

```
#pragma once
#include "Command.h"

class CommandRight: public Command{
private:
    Commands command;
public:
```

```
    CommandRight();
    CommandRight(Commands command_user);
    [[nodiscard]] Commands get_instruction() const override;
    void change(Commands instruction) override;
    char execute() override;
};
```

Название файла: CommandRight.cpp

```
#include "CommandRight.h"

CommandRight::CommandRight() {
    this->command = Commands::Right;
}

CommandRight::CommandRight(Commands command_user) {
    this->command = command_user;
}

Commands CommandRight::get_instruction() const {
    return this->command;
}

char CommandRight::execute() {
    return 'D';
}

void CommandRight::change(Commands instruction) {
    this->command = instruction;
}
```

Название файла: Devises.h

```
#pragma once
#include "Command.h"

class Devices{
public:
    virtual Commands getCommand(char direction) = 0;
    virtual ~Devices(){

    }
};
```

Название файла: Console.h

```
#pragma once

#include <map>
#include "Devices.h"
```

```
class Console: public Devices{
private:
    std::map<char, Commands>dir;
public:
    Console();
    Commands getCommand(char direction) override;
};
```

Название файла: Console.cpp

```
#include "Console.h"


Commands Console::getCommand(char direction) {
    return dir[direction];
}

Console::Console() {
    dir['W'] = Commands::Up;
    dir['S'] = Commands::Down;
    dir['A'] = Commands::Left;
    dir['D'] = Commands::Right;
}
```

Название файла: Invoker.h

```
#pragma once
//#include "Command.h"
#include "Commands/CommandUp.h"
#include "Commands/CommandDown.h"
#include "Commands/CommandLeft.h"
#include "Commands/CommandRight.h"
#include "Devices.h"
#include "Console.h"
//#include "../Tools/Game.h"



class Invoker{
private:
//    std::map<char, Commands>dir;
    //Console console;
    Devices* devices;
    Command *Up;
    Command *Down;
    Command *Left;
    Command *Right;
public:
    Invoker();
```

```
            void SetCommand(Command* up, Command* down, Command*
left, Command* right);
      //    void executeCommandUp();
      //    void executeCommandDown();
      //    void executeCommandLeft();
      //    void executeCommandRight();
          char run();
          ~Invoker();
      };

      //void  Invoker::SetCommand(Command*  up,  Command*  down,
Command* left, Command* right) {
      //    this->Up = dynamic_cast<CommandUp*>(up);
      //    this->Down = dynamic_cast<CommandDown*>(down);
      //    this->Left = dynamic_cast<CommandLeft*>(left);
      //    this->Right = dynamic_cast<CommandRight*>(right);
      //    this->devices = new Console();
      ////    dir['W'] = Commands::Up;
      ////    dir['S'] = Commands::Down;
      ////    dir['A'] = Commands::Left;
      ////    dir['D'] = Commands::Right;
      //}
      //
      //char Invoker::run() {
      //    char command;
      //    std::cout << "Enter command: ";
      //    std::cin >> command;
      //    std::cout << std::endl;
      //
      //                  if    (devices->getCommand(command)    ==
this->Up->get_instruction()){  // dir[command]
      //        return this->Up->execute();
      //    }
      //             else   if   (devices->getCommand(command)   ==
this->Down->get_instruction()){
      //        return this->Down->execute();
      //    }
      //             else   if   (devices->getCommand(command)   ==
this->Right->get_instruction()){
      //        return this->Right->execute();
      //    }
      //             else   if   (devices->getCommand(command)   ==
this->Left->get_instruction()){
      //        return this->Left->execute();
      //    }
      //    else{
      //        std::cout << "Default: UP" << std::endl;
      //        return this->Up->execute();
      //    }
      //
      //
```

```
//}


//void Invoker::executeCommandUp() {
//    // нужна проверка
//    this->Up->execute();
//}
//
//void Invoker::executeCommandDown() {
//    this->Down->execute();
//}
//
//void Invoker::executeCommandLeft() {
//    this->Left->execute();
//}
//
//void Invoker::executeCommandRight() {
//    this->Right->execute();
//}
```

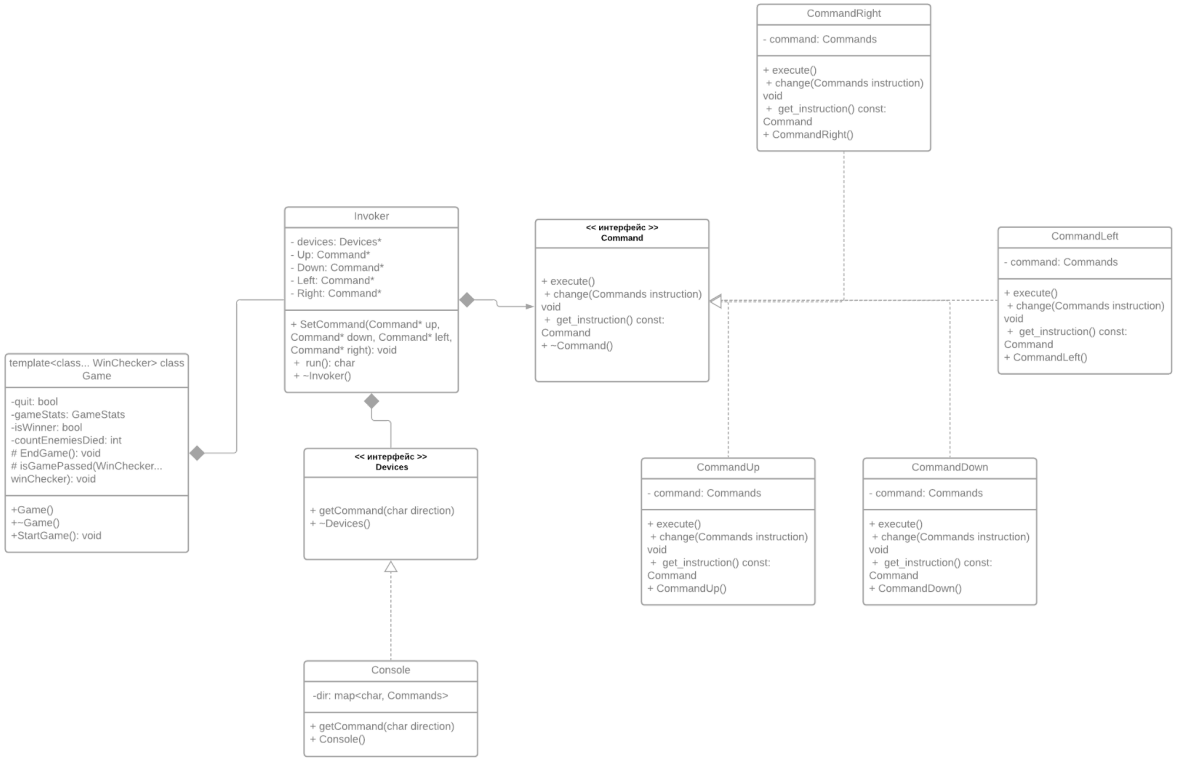
Название файла: Invoker.cpp

```
#include "Invoker.h"
#include <iostream>

void Invoker::SetCommand(Command* up, Command* down, Command*
left, Command* right) {
    this->Up = dynamic_cast<CommandUp*>(up);
    this->Down = dynamic_cast<CommandDown*>(down);
    this->Left = dynamic_cast<CommandLeft*>(left);
    this->Right = dynamic_cast<CommandRight*>(right);
    this->devices = new Console();
//    dir['W'] = Commands::Up;
//    dir['S'] = Commands::Down;
//    dir['A'] = Commands::Left;
//    dir['D'] = Commands::Right;
}

char Invoker::run() {
    char command;
    std::cout << "Enter command: ";
    std::cin >> command;
    std::cout << std::endl;

                    if     (devices->getCommand(command)     ==
this->Up->get_instruction()){  // dir[command]
        return this->Up->execute();
    }
```

```
                    else    if    (devices->getCommand(command)    ==
this->Down->get_instruction()){
                return this->Down->execute();
            }
                    else    if    (devices->getCommand(command)    ==
this->Right->get_instruction()){
                return this->Right->execute();
            }
                    else    if    (devices->getCommand(command)    ==
this->Left->get_instruction()){
                return this->Left->execute();
            }
            else{
                std::cout << "Default: UP" << std::endl;
                return this->Up->execute();
            }


    }

    Invoker::Invoker() {
        devices = nullptr;
        Up = nullptr;
        Down = nullptr;
        Left = nullptr;
        Right = nullptr;
    }

    Invoker::~Invoker() {
        delete Up;
        delete Down;
        delete Left;
        delete Right;
        delete devices;
    }
```

Приложение Б

UML-диаграмма классов


CommandRight
- command: Commands
+ execute()
+ change(Commands instruction) void
+ get_instruction() const: Command
+ CommandRight()
Invoker
- devices: Devices*
- Up: Command*
- Down: Command*
- Left: Command*
- Right: Command*
+ SetCommand(Command* up, Command* down, Command* left, Command* right): void
+ run(): char
+ ~Invoker()
<< интерфейс >>
Command
+ execute()
+ change(Commands instruction) void
+ get_instruction() const: Command
+ ~Command()
CommandLeft
- command: Commands
+ execute()
+ change(Commands instruction) void
+ get_instruction() const: Command
+ CommandLeft()
template<class... WinChecker> class Game
-quit: bool
-gameStats: GameStats
-isWinner: bool
-countEnemiesDied: int
EndGame(): void
isGamePassed(WinChecker... winChecker): void
+Game()
+~Game()
+StartGame(): void
<< интерфейс >>
Devices
+ getCommand(char direction)
+ ~Devices()
CommandUp
- command: Commands
+ execute()
+ change(Commands instruction) void
+ get_instruction() const: Command
+ CommandUp()
CommandDown
- command: Commands
+ execute()
+ change(Commands instruction) void
+ get_instruction() const: Command
+ CommandDown()
Console
-dir: map<char, Commands>
+ getCommand(char direction)
+ Console()

[Link]