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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «ООП»

ТЕМА: ДОБАВЛЕНИЕ ИГРОКА И ЭЛЕМЕНТОВ ПОЛЯ

Студент гр. 9383 _____ Гордон Д.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Создан класс игрока, которым управляет пользователь. Объект класса игрока может перемещаться по полю, а также взаимодействовать с элементами поля. Для элементов поля должен быть создан общий интерфейс и должны быть реализованы 3 разных класса элементов, которые по-разному взаимодействуют с игроком. Для взаимодействия игрока с элементом должен использоваться перегруженный оператор *(Например, оператор +)*. Элементы поля могут добавлять очки игроку/замедлять передвижения/и т.д.

Обязательные требования:

- Реализован класс игрока
- Реализованы три класса элементов поля
- Объект класса игрока появляется на клетке со входом
- Уровень считается пройденным, когда объект класса игрока оказывается на клетке с выходом (и при определенных условиях: например, набрано необходимое кол-во очков)
- Взаимодействие с элементами происходит через общий интерфейс
- Взаимодействие игрока с элементами происходит через перегруженный оператор

Дополнительные требования:

- Для создания элементов используется паттерн **Фабричный метод/Абстрактная фабрика**

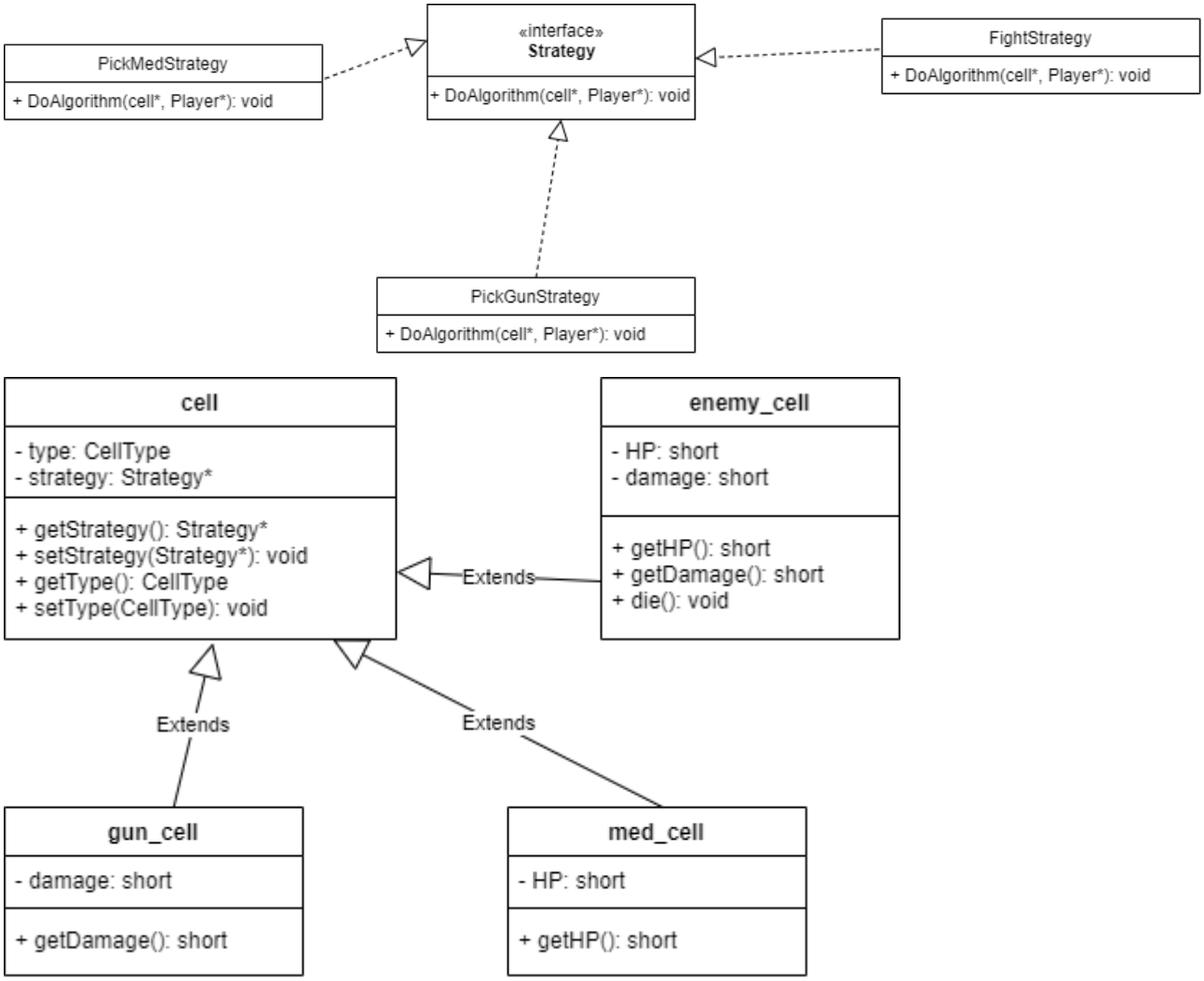
Реализовано динамическое изменение взаимодействия игрока с элементами через паттерн **Стратегия**. Например, при взаимодействии с определенным количеством элементов, игрок не может больше с ними взаимодействовать

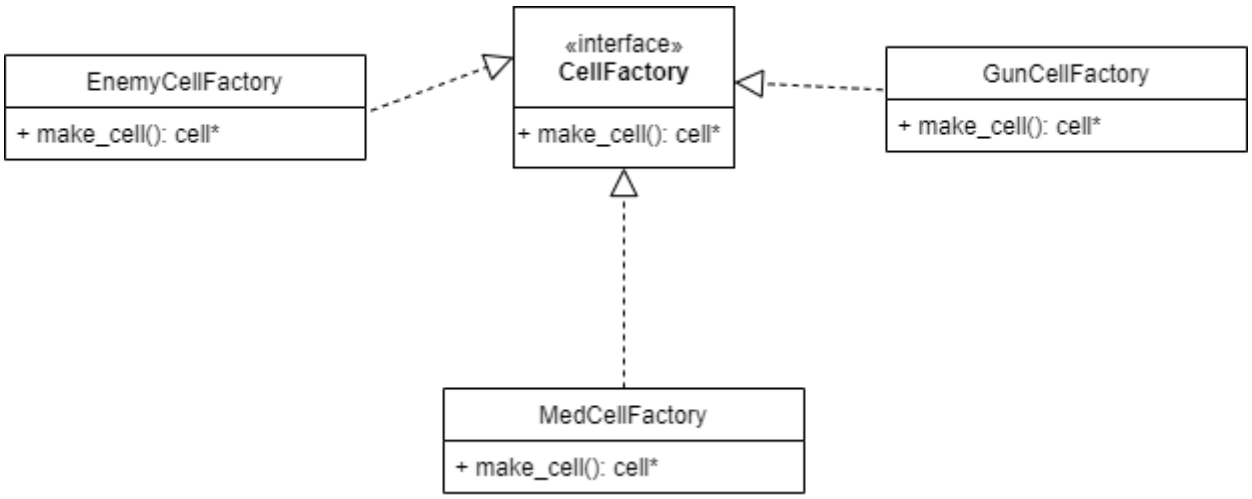
Основные теоретические положения.

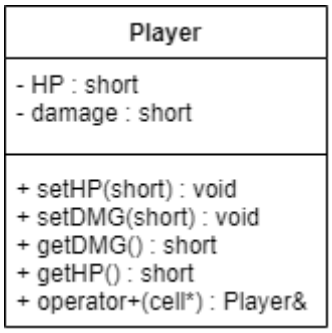
Фабрика – паттерн проектирования для создания объектов определенного типа (фабрика стульев и т.д.)

Стратегия – паттерн проектирования для поведения объектов.

UML диаграммы.




EnemyCellFactory
+ make_cell(): cell*
«interface»
CellFactory
+ make_cell(): cell*
GunCellFactory
+ make_cell(): cell*
MedCellFactory
+ make_cell(): cell*


Player
- HP : short
- damage : short
+ setHP(short) : void
+ setDMG(short) : void
+ getDMG() : short
+ getHP() : short
+ operator+(cell*) : Player&

Описание архитектурных решений.

Мною было принято решение создать несколько типов клеток(gun_cell, med_cell, enemy_cell) – три разных элемента поля. Затем создать стратегии для взаимодействия с ними.

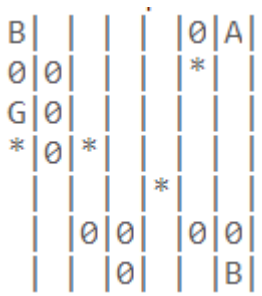
Создание клеток типа gun_cell, med_cell, enemy_cell происходит с помощью фабрики этих типов. Это происходит в методе initialize класса board.

Взаимодействие происходит через стратегии, а стратегия хранится в каждой клетке, вызов стратегии происходит в перегруженном операторе + класса Player.

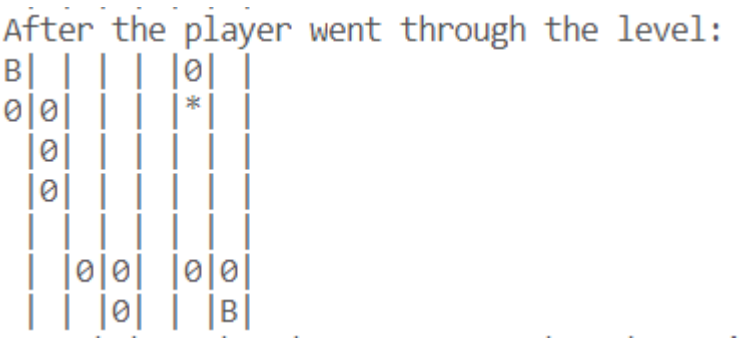
У Player есть очки здоровья и урон. В начале 90 и 0 соответственно. При взаимодействии с med_cell ему прибавляется 30 ОЗ, при взаимодействии с enemy_cell отнимается 30. Если Player взаимодействовал с gun_cell, то урон у него станет 30.

Результат работы программы.

До взаимодействия с клетками:



После взаимодействия:



Выводы.

Узнал, что такое фабричный метод, стратегия и научился их применять на практике. Научился работать с интерфейсами.

КОД ПРОГРАММЫ

Strategy.h:

```
#pragma once


class cell;
class Player;

class Strategy
{
public:
    virtual ~Strategy() {}
    virtual void DoAlgorithm(cell* Cell, Player* player) = 0;
};
```

PickMedStrategy.:

```
#pragma once

#include "Player.h"
#include "med_cell.h"

class PickMedStrategy : public Strategy
{
    void DoAlgorithm(cell* medcell, Player* player) override
    {
        //med_cell* ptr = new med_cell();
        med_cell* ptr = (med_cell*)medcell;
        player->setHP(player->getHP() + ptr->getHP());
        delete ptr->getStrategy();
        ptr->setStrategy(nullptr);
        medcell->setType(ORDINARY);
    }
};
```

PickGunStrategy.h:

```
#pragma once

#include "Player.h"
#include "gun_cell.h"

class PickGunStrategy : public Strategy
```

```
{
public:
    void DoAlgorithm(cell* guncell, Player* player) override
    {
        //gun_cell* ptr = new gun_cell();
        gun_cell* ptr = (gun_cell*)guncell;
        player->setDMG(ptr->getDamage());
        delete ptr->getStrategy();
        ptr->setStrategy(nullptr);
        guncell->setType(ORDINARY);
    }
};
```

FightStrategy.h:

```
#pragma once

#include "Strategy.h"
#include "enemy_cell.h"

class FightStrategy : public Strategy
{
public:
    void DoAlgorithm(cell* enemycell, Player* player) override
    {
        //enemy_cell* ptr = new enemy_cell();
        enemy_cell* ptr = (enemy_cell*)enemycell;
        player->setHP(player->getHP() - ptr->getDamage());
        if(player->getDMG() != 0)
        {
            ptr->die();
            delete ptr->getStrategy();
            ptr->setStrategy(nullptr);
            enemycell->setType(ORDINARY);
        }
    }
};
```

Cell.h:

```
#pragma once

#include <iostream>
#include "Strategy.h"

enum CellType{ ORDINARY, ENTRANCE, BLOCK, ENEMY, GUN, MED, EXIT}; /
*обычная, вход, выход*/

class cell
{
private:
    CellType type;
    Strategy* strategy;
public:
    cell()
    {
        type = ORDINARY;
        strategy = nullptr;
    }
    Strategy* getStrategy();
    void setStrategy(Strategy* strategy);
    CellType getType();
    void setType(CellType type);
};
```

Cell.cpp:

```
#include "include/cell.h"


void cell::setType(CellType type)
{
    this->type = type;
}

CellType cell::getType()
{
    return this->type;
}
```

```
Strategy* cell::getStrategy()
{
    return this->strategy;
}

void cell::setStrategy(Strategy* strategy)
{
    this->strategy = strategy;
}
```

Enemy_cell.h:

```
#pragma once

#include "cell.h"

class enemy_cell:public cell
{
public:
    enemy_cell()
    {
        setType(ENEMY);
        HP = 30;
        damage = 30;
    }
    short getHP();
```

```
    short getDamage();
    void die();
private:
    short HP;
    short damage;
};
```

Enemy_cell.cpp:

```
#include "include/enemy_cell.h"

short enemy_cell::getHP()
{
    return this->HP;
}

short enemy_cell::getDamage()
{
    return this->damage;
}

void enemy_cell::die()
{
    this->damage = 0;
    this->HP = 0;
}
```

Gun_cell.h:

```
#pragma once

#include "cell.h"

class gun_cell:public cell
{
public:
    gun_cell()
    {
        setType(GUN);
        damage = 30;
    }
    short getDamage();
private:
```

```
    short damage;
};
```

Gun_cell.cpp:

```
#include "include/gun_cell.h"
short gun_cell::getDamage()
{
    return this->damage;
}
```

Med_cell.h:

```
#pragma once


#include "cell.h"

class med_cell:public cell
{
public:
    med_cell()
    {
        setType(MED);
        HP = 30;
    }
    short getHP();
```

```
private:
    short HP;
};
```

MedCell.cpp:

```
#include "include/med_cell.h"

short med_cell::getHP()
{
    return this->HP;
}
```

CellFactory.h:

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

class CellFactory
{
public:
    virtual cell* make_cell() = 0;
    virtual ~CellFactory() = default;
};
```

EnemyCellFactory.h:

```
#pragma once

#include "enemy_cell.h"
#include "CellFactory.h"

class EnemyCellFactory : CellFactory
{
public:
    cell* make_cell() override
    {
        return new enemy_cell();
    }
};
```

GunCellFactory.h:

```
#pragma once

#include "gun_cell.h"
#include "CellFactory.h"

class GunCellFactory : CellFactory
{
public:
    cell* make_cell() override
    {
        return new gun_cell();
    }
};
```

MedCellFactory.h:

```
#pragma once

#include "med_cell.h"
#include "CellFactory.h"

class MedCellFactory : CellFactory
{
public:
    cell* make_cell() override
    {
        return new med_cell();
    }
};
```

Board.h:

```
#pragma once

#include "iterator.h"
#include "Player.h"


#define ROW 7
#define COL 7

class board
{
private:
    board() = default;
    cell* arr[ROW * COL];
    Player player;
    //operators
    board(const board& other); //copy constructor

    board(board&& other) noexcept; //move constructor

    board& operator=(const board& other); //copy operator

    board& operator=(board&& other) noexcept; //move operator

public:
    static board& GetInstance();
```

```
    void print();
    void initialize();
    iterator<cell*> begin();
    iterator<cell*> end();
    void gameLogic();
};
```

Board.cpp:

```
#include "include/board.h"
#include "include/GunCellFactory.h"
#include "include/MedCellFactory.h"
#include "include/EnemyCellFactory.h"
#include "include/FightStrategy.h"
#include "include/PickGunStrategy.h"
#include "include/PickMedStrategy.h"

board& board::GetInstance()
{
    static board instance;
    return instance;
}

void board::print()
{
    int line = 0;
    for (iterator<cell*> itr = GetInstance().begin(); itr != GetIns
tance().end(); itr++)
    {
        if((*itr)->getType() == ENTRANCE)
            std::cout << "B";
        else if((*itr)->getType() == EXIT)
            std::cout << "B";
        else if((*itr)->getType() == GUN)
            std::cout << "G";
        else if((*itr)->getType() == MED)
            std::cout << "A";
        else if((*itr)->getType() == BLOCK)
            std::cout << "0";
        else if((*itr)->getType() == ENEMY)
            std::cout << "*";
```

```
        else
            std::cout << " ";
        std::cout << "|";
        line++;
        if (line % 7 == 0)
            std::cout << "\n";
    }
}

void board::initialize()
{
    static EnemyCellFactory enemies;
    static GunCellFactory guns;
    static MedCellFactory meds;
    //Entrance
    arr[0] = new cell();
    arr[0]->setType(ENTRANCE);
    //Exit
    arr[ROW*COL - 1] = new cell();
    arr[ROW*COL - 1]->setType(EXIT);
    //Block
    arr[5] = new cell();
    arr[5]->setType(BLOCK);
    arr[7] = new cell();
    arr[7]->setType(BLOCK);
    arr[8] = new cell();
    arr[8]->setType(BLOCK);
    arr[15] = new cell();
    arr[15]->setType(BLOCK);
    arr[22] = new cell();
    arr[22]->setType(BLOCK);
    arr[37] = new cell();
    arr[37]->setType(BLOCK);
    arr[38] = new cell();
    arr[38]->setType(BLOCK);
    arr[40] = new cell();
    arr[40]->setType(BLOCK);
    arr[41] = new cell();
    arr[41]->setType(BLOCK);
    arr[45] = new cell();
    arr[45]->setType(BLOCK);
```

```
    //Enemies
    arr[12] = enemies.make_cell();
    arr[12]->setStrategy(new FightStrategy);
    arr[21] = enemies.make_cell();
    arr[21]->setStrategy(new FightStrategy);
    arr[23] = enemies.make_cell();
    arr[23]->setStrategy(new FightStrategy);
    arr[32] = enemies.make_cell();
    arr[32]->setStrategy(new FightStrategy);
    //GUN
    arr[14] = guns.make_cell();
    arr[14]->setStrategy(new PickGunStrategy);
    //MED
    arr[6] = meds.make_cell();
    arr[6]->setStrategy(new PickMedStrategy);
    for(int i = 1; i < ROW*COL - 1; i++)
    {
        if(arr[i] == nullptr)
            arr[i] = new cell();
    }
}

iterator<cell*> board::begin()
{
    return iterator(arr);
}

iterator<cell*> board::end()
{
    return iterator(arr + ROW * COL);
}

void board::gameLogic()
{
    for(iterator itr = GetInstance().begin(); itr != GetInstance().
end(); itr++)
    {
        player + *itr;
    }
}
```

```
//operators

board::board(const board& other) //copy constructor
{
    for (int i = 0, j = 0; i < ROW * COL; ++i, ++j)
        this->arr[i] = other.arr[j];
}

board::board(board&& other) noexcept //move constructor
{
    for (int i = 0, j = 0; i < ROW * COL; ++i, ++j)
            this->arr[i] = other.arr[j];
        for (int i = 0; i < ROW * COL; ++i)
        {
            other.arr[i]->setType(ORDINARY);
        }
}

board& board::operator=(const board& other) //copy operator
{
    if (this == &other)
            return *this;
        for (int i = 0, j = 0; i < ROW * COL; ++i, ++j)
            this->arr[i] = other.arr[j];
        return *this;
}

board& board::operator=(board&& other) noexcept //move operator
{
    if (this == &other)
            return *this;
        for (int i = 0, j = 0; i < ROW * COL; ++i, ++j)
            this->arr[i] = other.arr[j];
        for (int i = 0; i < ROW * COL; ++i)
        {
            //other.arr[i] = cell(); ?
            other.arr[i]->setType(ORDINARY);
        }
        return *this;
}
```

Player.h:

```
#pragma once

#include "cell.h"

class Player
{
public:
    Player()
    {
        this->HP = 90;
        this->damage = 0;
    }
    ~Player()
    {
        this->HP = 0;
        this->damage = 0;
    }
    Player& operator+(cell* Cell);
    void setHP(short HP);
    void setDMG(short dmg);
    short getDMG();
    short getHP();
private:
    short HP;
    short damage;
};
```

Player.cpp:

```
#include "include/player.h"

void Player::setHP(short HP)
{
    this->HP = HP;
}

void Player::setDMG(short dmg)
{
    this->damage = dmg;
}
```

```
short Player::getDMG()
{
    return this->damage;
}

short Player::getHP()
{
    return this->HP;
}

//operator+
Player& Player::operator+(cell* Cell)
{
    if(Cell->getStrategy() != nullptr)
    {
        Cell->getStrategy()->DoAlgorithm(Cell, this);
    }
    return *this;
}
```

Source.cpp:

```
#include "include/board.h"

int main()
{
    auto& Board = board::GetInstance();
    Board.initialize();
    Board.print();
    Board.gameLogic();
    std::cout << "After the player went through the level: \n";
    Board.print();
    return 0;
}
```