

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

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Объектно-ориентированное программирование»
Тема: "Добавление игрока и элементов для поля"

Студент гр. 9383

Корсунов А.А.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Написать программу в ООП стиле согласно заданию. Углубить знания об ООП, по возможности изучить предложенные паттерны.

Задание.

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

Выполнение работы.

Для выполнения работы использовалась библиотека SFML, предназначенная для работы с 2D графикой.

Класс Cell:

Данный класс был обновлен. Были добавлены булевы свойства игрока и элементов взаимодействия с ними, свойство класса `Element*`, хранящий указатель на созданный элемент взаимодействия и свойство класса `Player_1*`, хранящий указатель на созданный элемент игрока. Были добавлены новые `set()`-ры и `get()`-ры для новых свойств, а также два метода `delete()`, которые подчищают созданные элементы. Теперь от этого класса наследуется класс игрока и элемента.

Класс Field:

Данный класс был обновлен. Были добавлены два метода, которые устанавливают игрока в клетку и удаляют его из нее. Был объявлен дружественный класс итератор. Теперь от этого класса наследуется класс игрока.

Класс Player_1:

Данный класс является новым. Это класс игрока, у него шесть свойств, отвечающие за количество здоровья у игрока, брони, опьянения (класс к этому свойству еще не доработан), превращения в монстра (класс к этому свойству еще не создан), количество спасенных людей. Был определен конструктор для определения полей. Созданы `get()`-ры для них. Созданы методы прибавления и убавления свойств на единицу (кроме свойства спасенных людей). Так же определены свойства координат и метод их изменения, а также `get()`-ры к ним.

Элементы поля были реализованы с помощью паттерна «Фабричный метод».

Класс Element:

Данный класс создан, чтобы перегружать виртуальную функцию перегрузки оператора+.

Класс Medicine:

Данный класс создан, чтобы через перегруженный оператор+ добавлять здоровья игроку.

Класс Armor:

Данный класс создан, чтобы через перегруженный оператор+ добавлять броню игроку.

Класс People:

Данный класс создан, чтобы через перегруженный оператор+ добавлять спасенных объектов игроку.

Класс Alcogol:

Данный класс создан, чтобы через перегруженный оператор+ добавлять очки опьянения игроку.

Класс Enemy:

Данный класс создан, чтобы через перегруженный оператор+ отнимать здоровье игроку.

Все элементы исчезают после взаимодействия с игроком.

Были созданы классы абстрактной фабрики с виртуальной функцией, которая возвращает указатель на созданный элемент.

Класс Factory_Element:

Данный класс является шаблоном для других классов, с помощью которого создается и возвращается указатель на созданный элемент через виртуальную функцию.

Классы Factory_*, где * - все оставшиеся классы взаимодействия объектов:

Данные классы созданы для того, чтобы создавать элементы и возвращать на них указатели.

Класс drow:

Данный класс был очень сильно обновлен. От него наследуются фабричные классы, класс итератор и класс игрока. Добавлены свойства x и y, которые помогают определять координаты игрока на поле. Были полностью заменены текстуры всех объектов на поле (определения объектов будут подписаны под примером работы программы). Здесь же теперь создается объект класса Player_1. Добавлено рисование на поле состояния выигрыша (выигранной играется считается, если игрок «спас» всех кроликов на поле и добрался до финиша), состояние проигрыша (если количество очков здоровья равняется нулю) и в реальном времени рисуются свойства игрока. В этом же классе устанавливаются объекты взаимодействия на поле.

Класс Iterator:

Реализован стандартный паттерн итератор с прошлой лабораторной работы.



Рисунок 1 — Пример работы программы (рыцарь — игрок, лестница — вход, ворота — выход, кролик — объект, который нужно спасти, грибы повышают здоровья, гоблины — отнимают здоровье, щит — добавляет брони, фляга — добавляет очки опьянения, плиты и колонны соответственно проходимые и непроходимые клетки).

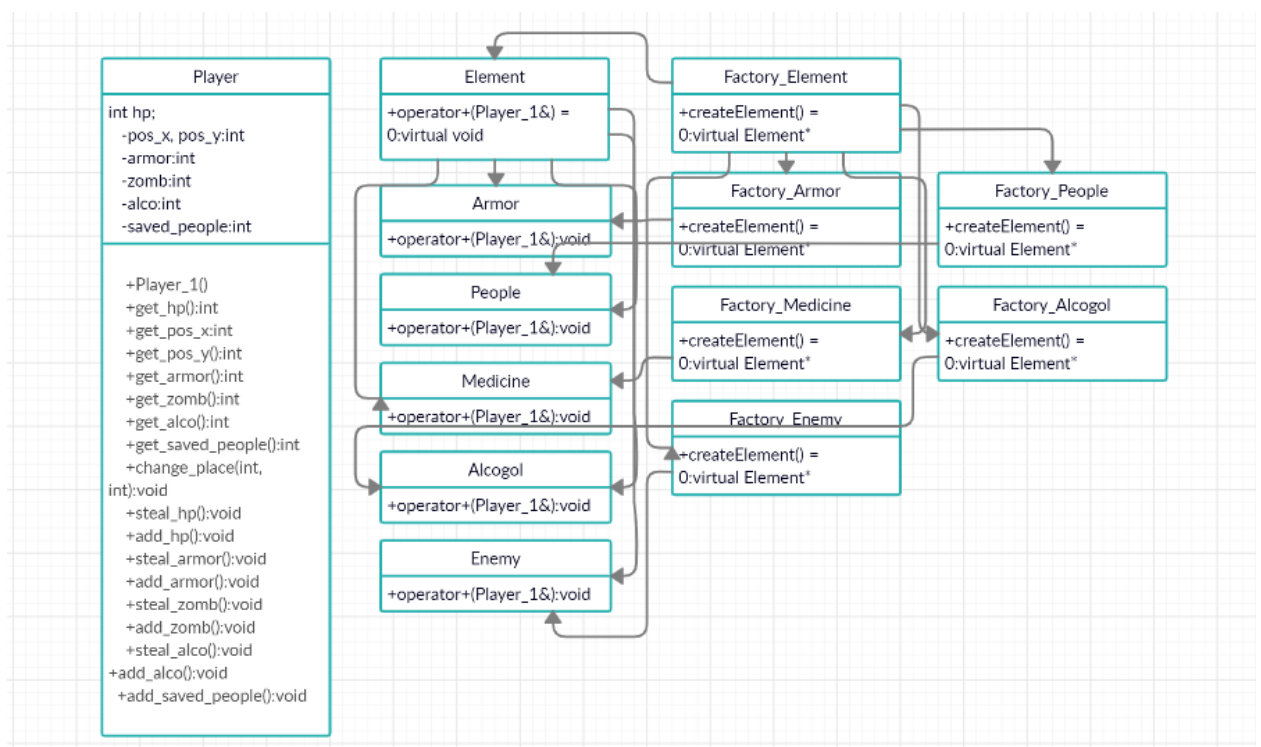


Рисунок 2.

На рисунке 2 изображена UML-диаграмма новых классов, реализованных в данной работе.

Выводы.

Написана программа в ООП стиле согласно заданию. Углублены знания об ООП, реализованы некоторые из паттернов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include "Cell.h"
#include "Field.h"
#include "drow.cpp"

using namespace std;

int main()
{
    Field* field = Field::get_Field(30, 10);
    for (int i = 2; i < 30; i++)
    {
        field->Unpass(i, 0, 0);
    }
    for (int i = 0; i < 8; i++)
    {
        field->Unpass(29, i, 0);
    }
    for (int i = 1; i < 10; i++)
    {
        field->Unpass(0, i, 0);
    }
    for (int i = 1; i < 29; i++)
    {
        field->Unpass(i, 9, 0);
    }
}
```



```

field->In(0,0,1);
field->Player_1(0,0,1);
field->Out(29,9,1);
field->Unpass(7,2,0);
field->Unpass(7,3,0);
field->Unpass(7,4,0);
field->Unpass(4,5,0);
field->Unpass(4,6,0);
field->Unpass(4,7,0);
field->Unpass(4,8,0);
field->Unpass(2,6,0);
field->Unpass(2,7,0);
field->Unpass(6,6,0);
field->Unpass(6,7,0);
field->Unpass(1,4,0);
field->Unpass(4,7,0);
for (int i = 9; i < 29; i++)
{
    field->Unpass(i, 2, 0);
}
field->Unpass(10,6,0);
for (int i = 11; i < 28; i++)
{
    field->Unpass(i, 6, 0);
}
field->Unpass(10,3,0);
field->Unpass(10,4,0);
field->Unpass(9,4,0);
field->Unpass(9,5,0);

```

```

        field->Unpass(9,6,0);
    field->Unpass(4,8,0);
    field->Unpass(2,4,0);
    field->Unpass(4,4,0);
    field->Unpass(6,5,0);
    field->Unpass(2,0,0);
    field->Unpass(2,1,0);
    field->Unpass(2,2,0);
    field->Unpass(3,2,0);
    field->Unpass(4,2,0);
    field->Unpass(5,2,0);
    field->Unpass(6,2,0);
    field->Unpass(6,4,0);
    draw go;
    go.show(field);
    return 0;
}

```

Файл Cell.h

```

#pragma once
#include "1_Player.h"
#include "Element.h"

```

```

class Cell
{
private:
    bool pass;
    bool out;
    bool in;
    bool player_1;

```

```
bool people;  
bool alcogol;  
bool enemy;  
bool medicine;  
bool armor;
```

```
Element* element;  
Player_1* player_11;
```

```
public:
```

```
Cell();  
~Cell();
```

```
void set_unpass(bool value);  
void set_out(bool value);  
void set_in(bool value);  
void set_player_1(bool value);  
void set_people(bool value);  
void set_alcogol(bool value);  
void set_enemy(bool value);  
void set_medicine(bool value);  
void set_armor(bool value);
```

```
bool get_pass();  
bool get_out();  
bool get_in();  
bool get_player_1();  
bool get_people();  
bool get_alcogol();
```

```

bool get_enemy();
bool get_medicine();
bool get_armor();

void set_Element(Element* elem);
void set_Player_1(Player_1* player);
void delete_Element();
void delete_Player_1();
Element* get_Element();
};

```

Файл Cell.cpp

```
#include "Cell.h"
```

```

Cell::Cell()
{
    this->pass = true;
    this->in = false;
    this->out = false;
    this->player_1 = false;
    this->element = nullptr;
    this->people = false;
    this->alcogol = false;
    this->armor = false;
    this->enemy = false;
    this->medicine = false;
    this->player_11 = nullptr;
}

```

```
Cell::~Cell(){};
```

```
void Cell::set_unpass(bool val)
{
    this->pass = val;
    this->in = false;
    this->out = false;
}
```

```
void Cell::set_in(bool val)
{
    this->pass = true;
    this->in = val;
    this->out = false;
}
```

```
void Cell::set_out(bool val)
{
    this->pass = true;
    this->in = false;
    this->out = val;
}
```

```
void Cell::set_player_1(bool val)
{
    this->player_1 = val;
}
```

```
void Cell::set_people(bool val)
{

```

```
    this->people = val;  
}
```

```
void Cell::set_alcogol(bool val)  
{  
    this->alcogol = val;  
}
```

```
void Cell::set_armor(bool val)  
{  
    this->armor = val;  
}
```

```
void Cell::set_enemy(bool val)  
{  
    this->enemy = val;  
}
```

```
void Cell::set_medicine(bool val)  
{  
    this->medicine = val;  
}
```

```
bool Cell::get_pass()  
{  
    return this->pass;  
}
```

```
bool Cell::get_in()
```

```
{  
    return this->in;  
}
```

```
bool Cell::get_out()  
{  
    return this->out;  
}
```

```
bool Cell::get_player_1()  
{  
    return this->player_1;  
}
```

```
void Cell::set_Element(Element* elem)  
{  
    this->element = elem;  
}
```

```
void Cell::set_Player_1(Player_1* player)  
{  
    this->player_1 = player;  
}
```

```
void Cell::delete_Element()  
{  
    if (this->element == nullptr)  
    {  
        return;  
    }
```

```

    }
    delete this->element;
    this->element = nullptr;
}

void Cell::delete_Player_1()
{
    if (this->player_11 == nullptr)
    {
        return;
    }
    delete this->player_11;
    this->player_11 = nullptr;
}

bool Cell::get_alcogol()
{
    return this->alcogol;
}

bool Cell::get_armor()
{
    return this->armor;
}

bool Cell::get_enemy()
{
    return this->enemy;
}

```



```
bool Cell::get_medicine()
{
    return this->medicine;
}
```

```
bool Cell::get_people()
{
    return this->people;
}
```

```
Element* Cell::get_Element()
{
    return this->element;
}
```

Файл Field.h

```
#pragma once
#include "Cell.h"
#include "1_Player.h"
```

```
class Field
{
private:
    Cell** ptr = nullptr;
    int width, height;
    static Field* object;

    Field(int width, int height);
    ~Field();
}
```

```

Field(const Field& ref_Field);
Field& operator=(const Field& ref_Field);
Field(Field&& ref_Field);
Field& operator=(Field&& ref_Field);

```

public:

```

static Field* get_Field(int x, int y);
void In(int x, int y, bool val);
void Out(int x, int y, bool val);
void Unpass(int x, int y, bool val);
void Player_1(int x, int y, bool val);
void del_Player_1(int x, int y, bool val);

```

```

friend class drow;
friend class Iterator;
};

```

Файл Field.cpp

```
#include "Field.h"
```

```
Field* Field::object = nullptr;
```

```

Field::Field(int x, int y) : width(x), height(y)
{
    this->ptr = new Cell* [this->width];
    for (int i = 0; i < this->width; i++)
    {
        this->ptr[i] = new Cell [this->height];
    }
}

```

```
Field::~~Field()
{
    for (int i = 0; i < this->width; i++)
    {
        delete[] this->ptr[i];
    }
    delete[] this->ptr;
}
```

```
Field* Field::get_Field(int x, int y)
{
    object = new Field(x, y);
    return object;
}
```

```
void Field::In(int x, int y, bool val)
{
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)
    {
        this->ptr[x][y].set_in(val);
    }
}
```

```
void Field::Out(int x, int y, bool val)
{
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)
    {
        this->ptr[x][y].set_out(val);
    }
}
```

```
    }  
}
```

```
void Field::Unpass(int x, int y, bool val)  
{  
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)  
    {  
        this->ptr[x][y].set_unpass(0);  
    }  
}
```

```
void Field::Player_1(int x, int y, bool val)  
{  
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)  
    {  
        this->ptr[x][y].set_player_1(1);  
    }  
}
```

```
void Field::del_Player_1(int x, int y, bool val)  
{  
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)  
    {  
        this->ptr[x][y].set_player_1(0);  
    }  
}
```

```
Field::Field(const Field& ref_Field)  
{
```

```

this->width = ref_Field.width;
this->height = ref_Field.height;
this->ptr = new Cell* [ref_Field.width];
for (int i = 0; i < ref_Field.width; i++)
{
    this->ptr[i] = new Cell[ref_Field.height];
    for (int j = 0; j < ref_Field.height; j++)
    {
        this->ptr[i][j] = ref_Field.ptr[i][j];
    }
}
}

```

```

Field& Field::operator=(const Field& ref_Field)

```

```

{
    if (&ref_Field == this)
    {
        return *this;
    }

```

```

    if (this != &ref_Field){
        for (int i = 0; i < this->width; i++)
        {
            delete[] this->ptr[i];
        }
        delete[] this->ptr;
    }

```

```

    this->width = ref_Field.width;

```

```

this->height = ref_Field.height;
this->ptr = new Cell* [ref_Field.width];
for (int i = 0; i < ref_Field.width; i++)
{
    this->ptr[i] = new Cell[ref_Field.height];
    for (int j = 0; j < ref_Field.height; j++)
    {
        this->ptr[i][j] = ref_Field.ptr[i][j];
    }
}
return *this;
}

```

```

Field::Field(Field&& ref_Field)
{
    this->ptr = ref_Field.ptr;
    this->width = ref_Field.width;
    this->height = ref_Field.height;
    ref_Field.ptr = nullptr;
    ref_Field.width = 0;
    ref_Field.height = 0;
}

```

```

Field& Field::operator=(Field&& ref_Field)
{
    if (&ref_Field == this)
    {
        return *this;
    }
}

```

```

if (this != &ref_Field){
    for (int i = 0; i < this->width; i++)
    {
        delete[] this->ptr[i];
    }
    delete[] this->ptr;
}

```

```

this->ptr = ref_Field.ptr;
this->width = ref_Field.width;
this->height = ref_Field.height;
ref_Field.ptr = nullptr;
ref_Field.width = 0;
ref_Field.height = 0;
return *this;
}

```

Файл drow.cpp

```

#pragma once
#include "Field.h"
#include <SFML/Graphics.hpp>
#include "FieldIterator.h"
#include "1_Player.h"
#include "Factory_Alcohol.h"
#include "Factory_Armor.h"
#include "Factory_Enemy.h"
#include "Factory_Medicine.h"
#include "Factory_People.h"

```

```

using namespace sf;

class drow
{
private:
    int x = 0;
    int y = 0;
    int w = 32;

public:
    void show(Field* field)
    {
        RenderWindow app(VideoMode(32 * field->width, 32 * (field-
>height+2)), "Cool game :)");

        Texture t;
        //t.loadFromFile("C:/Users/Eldorado/Documents/qwe/oop/govno/titles.j
pg");
        t.loadFromFile("C:/Users/Eldorado/Documents/qwe/oop/govno/fantasy
-tileset.png");
        Sprite s(t);
        Player_1 man;
        field->ptr[0][0].set_Player_1(&man);
        Font font;
        font.loadFromFile("19849.ttf");
        Text Player_1_info;
        Player_1_info.setFont(font);
        Player_1_info.setCharacterSize(20);
        Player_1_info.setColor(Color::Red);
    }
}

```



```
Player_1_info.setStyle(Text::Bold);  
Player_1_info.setPosition(Vector2f(0,(field->height)*32));
```

```
Factory_People factory_people;  
Factory_Alcohol factory_alcohol;  
Factory_Armor factory_armor;  
Factory_Enemy factory_enemy;  
Factory_Medicine factory_medicine;
```

```
field->ptr[3][1].set_Element(factory_people.createElement());  
field->ptr[3][1].set_people(1);  
field->ptr[28][1].set_Element(factory_people.createElement());  
field->ptr[28][1].set_people(1);  
field->ptr[1][8].set_Element(factory_people.createElement());  
field->ptr[1][8].set_people(1);  
field->ptr[6][3].set_Element(factory_enemy.createElement());  
field->ptr[6][3].set_enemy(1);  
field->ptr[3][8].set_Element(factory_enemy.createElement());  
field->ptr[3][8].set_enemy(1);  
field->ptr[7][5].set_Element(factory_enemy.createElement());  
field->ptr[7][5].set_enemy(1);  
field->ptr[1][5].set_Element(factory_medicine.createElement());  
field->ptr[1][5].set_medicine(1);  
field->ptr[9][3].set_Element(factory_medicine.createElement());  
field->ptr[9][3].set_medicine(1);  
field->ptr[7][7].set_Element(factory_armor.createElement());  
field->ptr[7][7].set_armor(1);  
field->ptr[10][5].set_Element(factory_armor.createElement());  
field->ptr[10][5].set_armor(1);
```

```
field->ptr[28][3].set_Element(factory_alcogol.createElement());  
field->ptr[28][3].set_alcogol(1);
```

```
while (app.isOpen())  
{  
    app.clear();  
    for (int i = 0; i < field->width; i++)  
    {  
        for (int j = 0; j < field->height; j++)  
        {  
            if (!field->ptr[i][j].get_in() && !field->ptr[i][j].get_out() &&  
field->ptr[i][j].get_pass())  
            {  
                s.setTextureRect(IntRect(0, 1 * w, w, w));  
                s.setPosition(i*w, j*w);  
                app.draw(s);  
                //проходимая  
            }  
            if (field->ptr[i][j].get_in())  
            {  
                s.setTextureRect(IntRect(5 * w, 1 * w, w, w));  
                s.setPosition(i*w, j*w);  
                app.draw(s);  
                //вход  
            }  
            if (field->ptr[i][j].get_out())
```

```

{
    s.setTextureRect(IntRect(1 * w, 3 * w, w, w));
    s.setPosition(i*w, j*w);
    app.draw(s);
    //ВЫХОД
}
if (!field->ptr[i][j].get_pass())
{
    s.setTextureRect(IntRect(0 * w, 3 * w, w, w));
    s.setPosition(i*w, j*w);
    app.draw(s);
    //непроходимая
}
if (field->ptr[i][j].get_player_1())
{
    s.setTextureRect(IntRect(5 * w, 18 * w, w, w));
    s.setPosition(i*w, j*w);
    app.draw(s);
}
if (field->ptr[i][j].get_people())
{
    s.setTextureRect(IntRect(1 * w, 20 * w, w, w));
    s.setPosition(i*w, j*w);
    app.draw(s);
}
if (field->ptr[i][j].get_enemy())
{
    s.setTextureRect(IntRect(0 * w, 18 * w, w, w));
    s.setPosition(i*w, j*w);

```

```

        app.draw(s);
    }
    if (field->ptr[i][j].get_medicine())
    {
        s.setTextureRect(IntRect(0 * w, 20 * w, w, w));
        s.setPosition(i*w, j*w);
        app.draw(s);
    }
    if (field->ptr[i][j].get_armor())
    {
        s.setTextureRect(IntRect(7 * w, 13 * w, w, w));
        s.setPosition(i*w, j*w);
        app.draw(s);
    }
    if (field->ptr[i][j].get_alcogol())
    {
        s.setTextureRect(IntRect(6 * w, 5 * w, w, w));
        s.setPosition(i*w, j*w);
        app.draw(s);
    }
}
}

```

```

if (man.get_hp() == 0)
{
    Player_1_info.setString("Game over");
}

```

```

        app.draw(Player_1_info);
    }
    else if (man.get_saved_people() == 3 && x == field->width-1 && y
== field->height-1)
    {
        Player_1_info.setString("Victory");
        app.draw(Player_1_info);
    }
    else
    {
        Player_1_info.setString("Hp: " + std::to_string(man.get_hp()) + "\
tarmor:  " + std::to_string(man.get_armor()) + "\tsaved  poeple:  " +
std::to_string(man.get_saved_people()) + "\talco:      " +
std::to_string(man.get_alco()));
        app.draw(Player_1_info);
    }
    Event e;

    while(app.pollEvent(e))
    {
        if (e.type == Event::Closed)
            app.close();
        if (e.type == Event::KeyPressed)
        {
            if (e.key.code == Keyboard::Escape) app.close();
            if (e.key.code == Keyboard::Left)
            {
                if ((x-1) >= 0)
                {

```

```

if (field->ptr[x-1][y].get_pass())
{
    x--;
    man.change_place(x, y);
    field->ptr[x][y].set_Player_1(&man);
    field->ptr[x+1][y].delete_Player_1();
    field->Player_1(x,y,1);
    field->del_Player_1(x+1,y,0);
    if (field->ptr[x][y].get_people())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_people(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_enemy())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_enemy(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_medicine())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_medicine(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_armor())
    {
        *(field->ptr[x][y].get_Element()) + man;

```

```

        field->ptr[x][y].set_armor(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_alcogol())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_alcogol(0);
        field->ptr[x][y].delete_Element();
    }
}
}
}
if (e.key.code == Keyboard::Right)
{
    if ((x+1) < field->width)
    {
        if (field->ptr[x+1][y].get_pass())
        {
            x++;
            man.change_place(x, y);
            field->ptr[x][y].set_Player_1(&man);
            field->ptr[x-1][y].delete_Player_1();
            field->Player_1(x,y,1);
            field->del_Player_1(x-1,y,0);
            if (field->ptr[x][y].get_people())
            {
                *(field->ptr[x][y].get_Element()) + man;
                field->ptr[x][y].set_people(0);
                field->ptr[x][y].delete_Element();
            }
        }
    }
}

```

```

    }
    if (field->ptr[x][y].get_enemy())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_enemy(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_medicine())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_medicine(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_armor())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_armor(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_alcogol())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_alcogol(0);
        field->ptr[x][y].delete_Element();
    }
}
}
}
if (e.key.code == Keyboard::Up)

```



```

{
    if ((y-1) >= 0)
    {
        if (field->ptr[x][y-1].get_pass())
        {
            y--;
            man.change_place(x, y);
            field->ptr[x][y].set_Player_1(&man);
            field->ptr[x][y+1].delete_Player_1();
            field->Player_1(x,y,1);
            field->del_Player_1(x,y+1,0);
            if (field->ptr[x][y].get_people())
            {
                *(field->ptr[x][y].get_Element()) + man;
                field->ptr[x][y].set_people(0);
                field->ptr[x][y].delete_Element();
            }
            if (field->ptr[x][y].get_enemy())
            {
                *(field->ptr[x][y].get_Element()) + man;
                field->ptr[x][y].set_enemy(0);
                field->ptr[x][y].delete_Element();
            }
            if (field->ptr[x][y].get_medicine())
            {
                *(field->ptr[x][y].get_Element()) + man;
                field->ptr[x][y].set_medicine(0);
                field->ptr[x][y].delete_Element();
            }
        }
    }
}

```

```

        if (field->ptr[x][y].get_armor())
        {
            *(field->ptr[x][y].get_Element()) + man;
            field->ptr[x][y].set_armor(0);
            field->ptr[x][y].delete_Element();
        }
        if (field->ptr[x][y].get_alcogol())
        {
            *(field->ptr[x][y].get_Element()) + man;
            field->ptr[x][y].set_alcogol(0);
            field->ptr[x][y].delete_Element();
        }
    }
}

if (e.key.code == Keyboard::Down)
{
    if ((y+1) < field->height)
    {
        if (field->ptr[x][y+1].get_pass())
        {
            y++;
            man.change_place(x, y);
            field->ptr[x][y].set_Player_1(&man);
            field->ptr[x][y-1].delete_Player_1();
            field->Player_1(x,y,1);
            field->del_Player_1(x,y-1,0);
            if (field->ptr[x][y].get_people())
            {

```

```

        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_people(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_enemy())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_enemy(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_medicine())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_medicine(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_armor())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_armor(0);
        field->ptr[x][y].delete_Element();
    }
    if (field->ptr[x][y].get_alcogol())
    {
        *(field->ptr[x][y].get_Element()) + man;
        field->ptr[x][y].set_alcogol(0);
        field->ptr[x][y].delete_Element();
    }
}

```

```

        }
    }
}
app.display();
}
}
};

```

Файл FieldIterator.h:

```

#pragma once
#include "Field.h"

class Iterator
{
private:
    int cell_x, cell_y;
    int width, height;
    const Field* field;

public:
    Iterator(const Field* f);
    Iterator(int i = 0, int j = 0);
    Iterator begin();
    Iterator end();

    void operator++();
    void operator--();
    bool operator==(const Iterator& field_2);

```

```

bool operator!=(const Iterator& field_2);
Cell& operator*();

Cell& get_Cell();

void next();
void back();
void up();
void down();
void left();
void right();
};

```

Файл FieldIterator.cpp:

```

#include "FieldIterator.h"

Iterator::Iterator(const Field* f)
{
    this->field = f;
    for (int i = 0; i < f->width; i++)
    {
        for (int j = 0; j < f->height; j++)
        {
            this->cell_x = i;
            this->cell_y = j;
        }
    }
}

```

```
Iterator::Iterator(int i, int j)
```

```
{  
    this->cell_x = i;  
    this->cell_y = j;  
}
```

```
Iterator Iterator::begin()
```

```
{  
    for (int i = 0; i < this->width; i++)  
    {  
        for (int j = 0; j < this->height; j++)  
        {  
            if (this->field->ptr[i][j].get_in())  
            {  
                return Iterator(i,j);  
            }  
        }  
    }  
}
```

```
Iterator Iterator::end()
```

```
{  
    for (int i = 0; i < this->width; i++)  
    {  
        for (int j = 0; j < this->height; j++)  
        {  
            if (this->field->ptr[i][j].get_out())  
            {  
                return Iterator(i,j);  
            }  
        }  
    }  
}
```

```

        }
    }
}

void Iterator::next()
{
    if ((this->cell_y + 1) == this->height && (this->cell_x + 1) == this-
>width)
    {
        return;
    }
    if ((this->cell_y + 1) < this->height)
    {
        this->cell_y++;
    }
    else
    {
        this->cell_x++;
    }
}

void Iterator::back()
{
    if ((this->cell_y + 1) == 1 && (this->cell_x + 1) == 1)
    {
        return;
    }
    if ((this->cell_y + 1) > 1)

```

```

        {
            this->cell_y--;
        }
        else
        {
            this->cell_x--;
        }
    }

    void Iterator::up()
    {
        if (this->cell_y > 0 && this->field->ptr[this->cell_x][this->cell_y -
1].get_pass())
        {
            this->cell_y--;
        }
    }

    void Iterator::down()
    {
        if (this->cell_y < this->height && this->field->ptr[this->cell_x][this-
>cell_y + 1].get_pass())
        {
            this->cell_y++;
        }
    }

    void Iterator::left()
    {

```



```

        if (this->cell_x > 0 && this->field->ptr[this->cell_x - 1][this-
>cell_y].get_pass())
        {
            this->cell_x--;
        }
    }

```

```

void Iterator::right()
{
    if (this->cell_x < this->width && this->field->ptr[this->cell_x + 1][this-
>cell_y].get_pass())
    {
        this->cell_x++;
    }
}

```

```

void Iterator::operator++()
{
    this->next();
}

```

```

void Iterator::operator--()
{
    this->back();
}

```

```

bool Iterator::operator==(const Iterator& field_2)
{

```

```

        return this->cell_x == field_2.cell_x && this->cell_y == field_2.cell_y
        && this->field == field_2.field;
    }

```

```

bool Iterator::operator!=(const Iterator& field_2)
{
    return this->cell_x != field_2.cell_x || this->cell_y != field_2.cell_y || this-
>field != field_2.field;
}

```

```

Cell& Iterator::get_Cell()
{
    return Field::object->ptr[this->cell_x][this->cell_y];
}

```

```

Cell& Iterator::operator*()
{
    return this->get_Cell();
}

```

Файл 1_Player.h:

```

#pragma once

```

```

class Player_1
{
private:
    int hp;
    int pos_x, pos_y;
    int armor;

```

```
int zomb;
int alco;
int saved_people;

public:
    Player_1();

    int get_hp();
    int get_pos_x();
    int get_pos_y();
    int get_armor();
    int get_zomb();
    int get_alco();
    int get_saved_people();

    void change_place(int, int);
    void steal_hp();
    void add_hp();
    void steal_armor();
    void add_armor();
    void steal_zomb();
    void add_zomb();
    void steal_alco();
    void add_alco();
    void add_saved_people();
};
```

Файл 1_Player.cpp:

```
#include "1_Player.h"
```

```
Player_1::Player_1()
```

```
{  
    this->hp = 3;  
    this->pos_x = 0;  
    this->pos_y = 0;  
    this->armor = 0;  
    this->zomb = 0;  
    this->alco = 0;  
    this->saved_people = 0;  
}
```

```
int Player_1::get_hp()
```

```
{  
    return this->hp;  
}
```

```
int Player_1::get_pos_x()
```

```
{  
    return this->pos_x;  
}
```

```
int Player_1::get_pos_y()
```

```
{  
    return this->pos_y;  
}
```

```
int Player_1::get_armor()
```

```
{  
    return this->armor;  
}
```

```
int Player_1::get_zomb()  
{  
    return this->zomb;  
}
```

```
int Player_1::get_alco()  
{  
    return this->alco;  
}
```

```
int Player_1::get_saved_people()  
{  
    return this->saved_people;  
}
```

```
void Player_1::change_place(int x, int y)  
{  
    this->pos_x++;  
    this->pos_y++;  
}
```

```
void Player_1::steal_hp()  
{  
    if (this->hp > 0)  
    {
```

```

        this->hp--;
    }
}

void Player_1::add_hp()
{
    if (this->hp < 3)
    {
        this->hp++;
    }
}

void Player_1::steal_armor()
{
    if (this->armor > 0)
    {
        this->armor--;
    }
}

void Player_1::add_armor()
{
    if (this->armor < 3)
    {
        this->armor++;
    }
}

void Player_1::steal_zomb()

```

```

{
    if (this->zomb > 0)
    {
        this->zomb--;
    }
}

void Player_1::add_zomb()
{
    if (this->zomb < 5)
    {
        this->zomb++;
    }
}

void Player_1::steal_alco()
{
    if (this->alco > 0)
    {
        this->alco--;
    }
}

void Player_1::add_alco()
{
    if (this->alco < 3)
    {
        this->alco++;
    }
}

```

```

}

void Player_1::add_saved_people()
{
    if (this->saved_people < 3)
    {
        this->saved_people++;
    }
}

```

Файл Element.h:

```

#pragma once
#include "1_Player.h"

class Element
{
public:
    virtual void operator+(Player_1&) = 0;
};

```

Файл Element.cpp:

```

#include "Element.h"

```

Файл Medicine.h:

```

#pragma once
#include "Element.h"

class Medicine:public Element
{

```



```
public:  
    void operator+(Player_1&);  
};
```

Файл Medicine.cpp:

```
#include "Medicine.h"  
  
void Medicine::operator+(Player_1& player_1)  
{  
    player_1.add_hp();  
}
```

Файл Armor.h:

```
#pragma once  
#include "Element.h"  
  
class Armor:public Element  
{  
public:  
    void operator+(Player_1&);  
};
```

Файл Armor.cpp:

```
#include "Armor.h"  
  
void Armor::operator+(Player_1& player_1)  
{  
    player_1.add_armor();  
}
```

Файл People.h:

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

class People:public Element
{
public:
    void operator+(Player_1&);
};
```

Файл People.cpp:

```
#include "People.h"

void People::operator+(Player_1& player_1)
{
    player_1.add_saved_people();
}
```

Файл Alcogol.h:

```
#pragma once
#include "Factory_Elements.h"
#include "Alcogol.h"

class Factory_Alcogol:public Factory_Element
{
public:
    Element* createElement();
};
```

Файл Alcogol.cpp:

```
#include "Factory_Alcogol.h"
```

```
Element* Factory_Alcogol::createElement()  
{  
    return new Alcogol;  
}
```

Файл Factory_Elements.h:

```
#pragma once
```

```
#include "Element.h"
```

```
class Factory_Element  
{  
public:  
    virtual Element* createElement() = 0;  
};
```

Файл Factory_Elements.cpp:

```
#include "Factory_Elements.h"
```

Файл Factory_Medicine.h:

```
#pragma once
```

```
#include "Factory_Elements.h"
```

```
#include "Medicine.h"
```

```
class Factory_Medicine:public Factory_Element
{
public:
    Element* createElement();
};
```

Файл Factory_Medicine.cpp:

```
#include "Factory_Medicine.h"
```

```
Element* Factory_Medicine::createElement()
{
    return new Medicine;
}
```

Файл Factory_People.h:

```
#pragma once
```

```
#include "Factory_Elements.h"
```

```
#include "People.h"
```

```
class Factory_People:public Factory_Element
{
public:
    Element* createElement();
};
```

Файл Factory_People.cpp:

```
#include "Factory_People.h"
```

```
Element* Factory_People::createElement()
```

```

{
    return new People;
}

```

Файл Factory_Enemy.h:

```

#pragma once
#include "Factory_Elements.h"
#include "Enemy.h"

class Factory_Enemy:public Factory_Element
{
public:
    Element* createElement();
};

```

Файл Factory_Enemy.cpp:

```

#include "Enemy.h"

void Enemy::operator+(Player_1& player_1)
{
    if (player_1.get_armor() == 0)
    {
        player_1.steal_hp();
    }
    else
    {
        player_1.steal_armor();
    }
}

```

Файл Factory_Armor.h:

```
#pragma once
#include "Factory_Elements.h"
#include "Armor.h"

class Factory_Armor:public Factory_Element
{
public:
    Element* createElement();
};
```

Файл Factory_Armor.cpp:

```
#include "Factory_Armor.h"

Element* Factory_Armor::createElement()
{
    return new Armor;
}
```

Файл Factory_Alcohol.h:

```
#pragma once
#include "Factory_Elements.h"
#include "Alcohol.h"

class Factory_Alcohol:public Factory_Element
{
public:
    Element* createElement();
```

```
};
```

Файл Factory_Alcohol.cpp:

```
#include "Factory_Alcohol.h"
```

```
Element* Factory_Alcohol::createElement()
```

```
{
```

```
    return new Alcohol;
```

```
}
```