

**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: Интерфейсы классов; взаимодействие классов; перегрузка**  
**операций**

Студентка гр. 8383

\_\_\_\_\_

Ишанина Л.Н.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2020

### **Цель работы.**

Разработать и реализовать набор интерфейсов классов, получить навыки в работе с ними и перегрузкой операций.

### **Ход работы.**

Разработать и реализовать набор классов:

- Класс базы
- Набор классов ландшафта карты
- Набор классов нейтральных объектов поля

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

- База должна размещаться на поле
- Методы для создания юнитов
- Учет юнитов, и реакция на их уничтожение и создание
- База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

- Должно быть создано минимум 3 типа ландшафта
- Все классы ландшафта должны иметь как минимум один интерфейс
- Ландшафт должен влиять на юнитов (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)
- На каждой клетке поля должен быть определенный тип ландшафта

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействовать юниты. Основные требования к классам нейтральных объектов поля:

- Создано не менее 4 типов нейтральных объектов

- Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций
- Классы нейтральных объектов должны иметь как минимум один общий интерфейс

### **Выполнение работы.**

#### **Класс базы(Base.h/Base.cpp)**

Был реализован класс Base, который наследуется от класса ObjectInterface и Observer. В поле базы хранятся указатели на 2 фабрики для создания юнитов, указатель на поле, массив, для хранения юнитов базы. Также класс Base содержит поля: здоровье базы и переменные-счетчики: номер базы, максимально количество юнитов, текущее количество юнитов.

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

- База должна размещаться на поле.

Размещение базы на поле происходит в классе Field, в методе appendElement(), представленном на рисунке 1. После того, как поле и база были созданы в main, вызывается данный метод и помещает базу в двумерный массив указателей на объекты. На рисунке 2,3 представлен демонстрационный пример размещения базы на поле.

```

//----- Добавление элемента на поле -----//
void Field::appendElement(unsigned newX, unsigned newY, ObjectInterface *unit) {
    if(!flag)
    {
        if (countObjects < controlMax && newX <= width && newY <= height)
        {
            if(obj[newX][newY] == nullptr)
            {
                countObjects++;
                obj[newX][newY] = unit;
            }
        }
    }
}
}

```

Рисунок 1 – метод добавления объекта на поле.

```

auto *field = new Field( height: 10, width: 10, controlMax: 5);

Unit** units = new Unit*[field->getControlMax()]; //выделяем место для массива
Base* base1 = new Base(field, units, ++counter, controlMax: 4);

field->appendElement( newX: 0, newY: 0, base1);
field->drawField();

```

Рисунок 2 – Запись в main.cpp

```

-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*|,*| |
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|#|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
-----

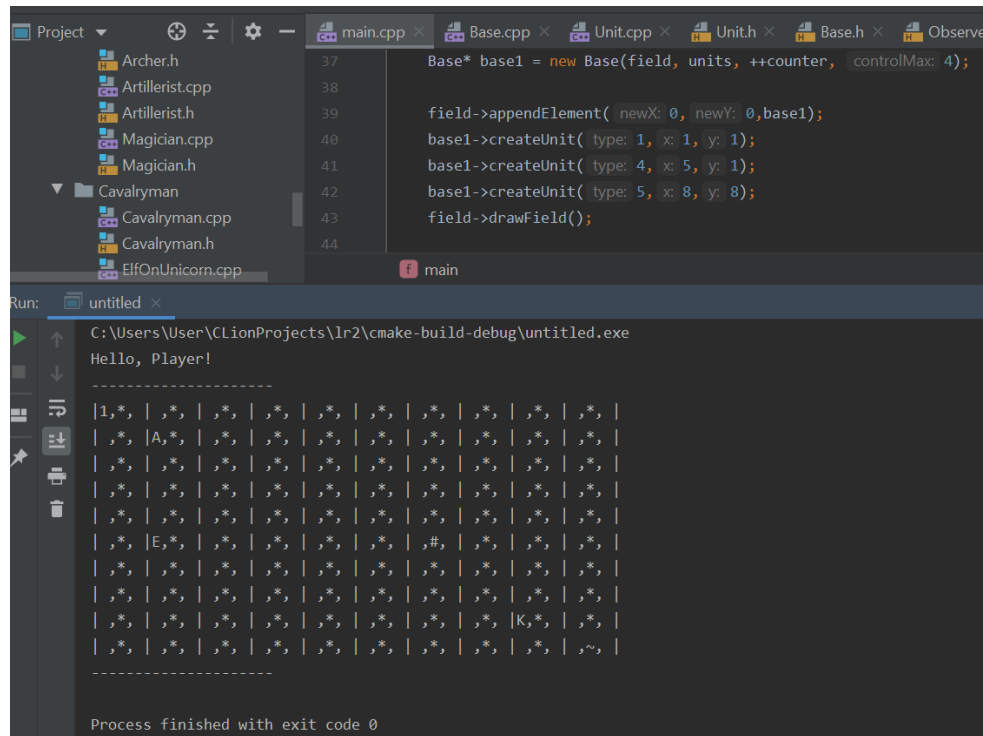
Process finished with exit code 0

```

Рисунок 3 – Вывод.

- Методы для создания юнитов.

База создает юнитов с помощью метода `createUnit()` (расположен в файле `Base.cpp`), который создает через фабрику юнита, записывает его в массив юнитов и сразу помещает на поле. На рисунке 4 показан демонстрационный пример работы создания юнитов через базу.



The screenshot shows a C++ IDE with a project named 'untitled'. The left sidebar displays a file tree with files like `Archer.h`, `Artillerist.cpp`, `Artillerist.h`, `Magician.cpp`, `Magician.h`, `Cavalryman` (a folder), `Cavalryman.cpp`, `Cavalryman.h`, and `ElfOnUnicorn.cpp`. The main editor shows the `main.cpp` file with the following code:

```
37 Base* base1 = new Base(field, units, ++counter, controlMax, 4);
38
39 field->addElement( newX: 0, newY: 0, base1);
40 base1->createUnit( type: 1, x: 1, y: 1);
41 base1->createUnit( type: 4, x: 5, y: 1);
42 base1->createUnit( type: 5, x: 8, y: 8);
43 field->drawField();
44
```

The bottom panel shows the output of the program:

```
Run: untitled x
C:\Users\User\CLionProjects\lr2\cmake-build-debug\untitled.exe
Hello, Player!
-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|A,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|E,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|,*|
-----
Process finished with exit code 0
```

Рисунок 4 – Вывод юнитов, созданных базой, на поле.

- Учет юнитов, и реакция на их уничтожение и создание.

База учитывает юнитов, так как она содержит массив юнитов, и при создании обязательно контролирует их число, чтобы не создавать юнитов больше, чем задано пользователем. База реагирует на уничтожение юнитов с помощью паттерна `Observer`. Метод `handleEvent` будет вызван из деструктора конкретного юнита при его уничтожении и далее будет уменьшен счетчик количества юнитов и массив юнитов будет сдвинут влево. Демонстрационный пример учета юнитов и реакции на их уничтожение и создание представлен на рисунке 5.

```

main.cpp
Project
  Archer.h
  Artillerist.cpp
  Artillerist.h
  Magician.cpp
  Magician.h
  Cavalryman
  Cavalryman.cpp
  Base.cpp
  Unit.cpp
  Unit.h
  Base.h
  Observer.h
  main.cpp
  42 base1->createUnit( type: 5, x: 8, y: 8);
  43 std::cout << "Count units: " << base1->getCountUnit() << std::endl;
  44 field->deleteUnit( newX: 8, newY: 8);
  45 std::cout << "Count units: " << base1->getCountUnit() << std::endl;
  46 field->drawField();
  47
  untitled
  Hello, Player!
  Count units: 3
  deleteUnit x=8 y=8
  Count units: 2
  -----
  |1,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|A,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|E,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  |,*|,*|,*|,*|,*|,*|,*|,*|,*|
  -----
  Process finished with exit code 0

```

Рисунок 5 – Демонстрационный пример учета юнитов и реакции на их уничтожение и создание.

- База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

В самом начале описания класса Base упоминается, какие поля содержатся в базе(рисунок 6), в том числе:

Здоровье, которому в конструкторе базы изначально приравнивается 500 и максимальное количество юнитов(int controlMax), которые могут быть одновременно созданы на базе. В данной реализации база может хранить столько юнитов, сколько пользователь сам задает. Также юниты могут повторяться, игра никак не ограничивает в создании, например, двух магов, возможно в дальнейшем будет введено ограничение на повторы.

```

class Base : public ObjectInterface, public Observer{
private:
    ConcreteFactoryMagicHero* fMagic;
    ConcreteFactoryStrongHero* fStrong;
    Field* field;
    int counter;//номер базы
    Unit** units;
    int countUnit;
    int controlMax;//максимальное количество юнитов
    int health;//здоровье

```

Рисунок 6 – поля класса Base.

### Класс ландшафта(Landscape.h)

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

- Должно быть создано минимум 3 типа ландшафта
- и
- Все классы ландшафта должны иметь как минимум один интерфейс

Изначально был создан класс-интерфейс ландшафта, в котором содержатся виртуальные методы: whatYouName() – метод, для отображения ландшафта на поле специальным символом, данный метод возвращает “имя” ландшафта (char name), сору() – метод копирования ландшафта и updateUnitHealth() – метод, который влияет на здоровье юнитов. От него наследуется класс ландшафта Landscape.h и уже от этого класса наследуются 3 типа ландшафта:

Пропась – Abyss.p/Abyss.cpp, на рисунке 7 показано содержание файла Abyss.cpp

```

#include "Abyss.h"
//пропасть

Abyss::Abyss() {
    name = '#';
}

char Abyss::whatYouName() {
    return name;
}

LandscapeInterface *Abyss::copy() {
    return new Abyss(*this);
}

void Abyss::updateUnitHealth(int* healthUnit, char nameUnit) {
    *healthUnit = 0;
}

```

Рисунок 7 – Abyss.cpp

Поляна – Glade.h/Glade.cpp, на рисунке 8 показано содержание файла Glade.cpp.

```

Glade::Glade() {
    name = '*';
}

char Glade::whatYouName() {
    return name;
}

LandscapeInterface *Glade::copy() {
    return new Glade(*this);
}

void Glade::updateUnitHealth(int* healthUnit, char nameUnit) {
}

```

Рисунок 8 – Glade.cpp

Болото – Swamp.h/Swamp.cpp, на рисунке 8 показано содержание файла Swamp.cpp



```

#include "Swamp.h"

Swamp::Swamp() {
    name = '~';
}

char Swamp::whatYouName() {
    return name;
}

LandscapeInterface *Swamp::copy() {
    return new Swamp(*this);
}

void Swamp::updateUnitHealth(int* healthUnit, char nameUnit) {
    *healthUnit = *healthUnit - 10;
}

```

Рисунок 9 – Swamp.cpp

- Ландшафт должен влиять на юнитов (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)

Ландшафт влияет на юнитов с помощью метода `updateUnitHealth()`, данный метод либо отнимает здоровье у юнитов, либо полностью убивает его. То есть в данной реализации, когда юнит передвигается на клетку поля с ландшафтом болото, то у него отнимается 10 очков от здоровья, если юнит ступает на клетку поля с ландшафтом пропасть, то он погибает.

Взаимодействие ландшафта с юнитами осуществляется с помощью паттерна Прокси. (Подробно про паттерн написано в пункте Дополнительные требования) Демонстрационные примеры влияния ландшафта на юнита представлены на рисунках 10,11,12.

```
field->appendElement( newX: 0, newY: 0,base1);
base1->createUnit( type: 1, x: 1, y: 1);
base1->createUnit( type: 4, x: 5, y: 1);
base1->createUnit( type: 5, x: 5, y: 5);
field->drawField();
field->move( x: 5, y: 5, newX: 5, newY: 6);
field->drawField();
```

Рисунок 10 – Вызываем метод перемещения юнита на клетку поля с ландшафтом пропасть в main.cpp

[illegible]

Рисунок 11 – Пример влияния ландшафта “пропасть” на юнита  
(юнит Knight убит)

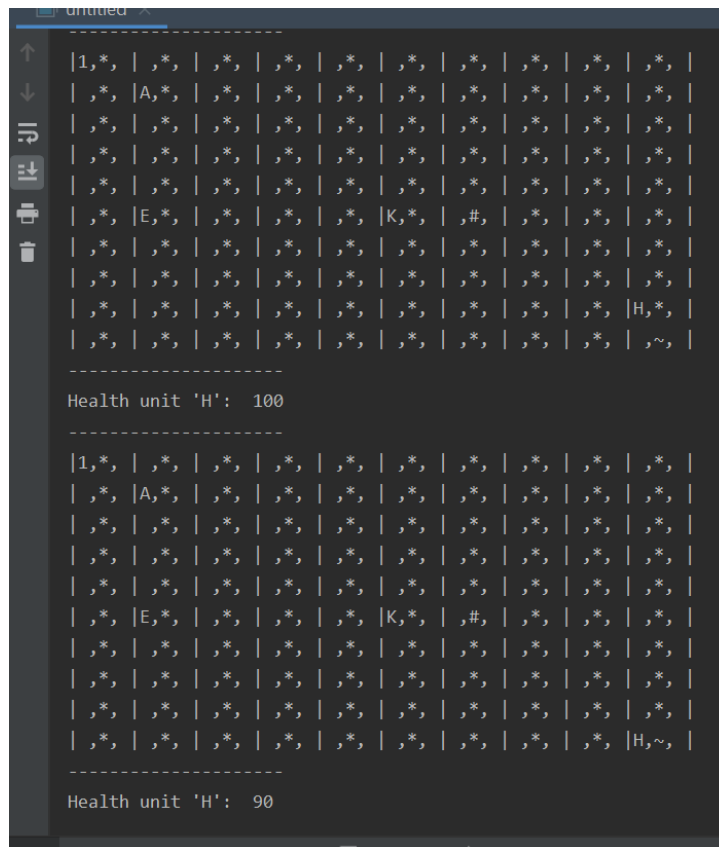


Рисунок 12 – Пример влияния ландшафта “болото” на здоровье юнита.

- На каждой клетке поля должен быть определенный тип ландшафта

Требование о том, что на каждой клетке поля должен быть ландшафт осуществляется в файле `Field.cpp` методом `setLandscapesOnField()`, который вызывается в конструкторе поля, выделяет память и автоматически заполняет клетки определенным ландшафтом. Демонстрационный пример ландшафта был показан на рисунках выше.

### Класс нейтральных объектов(NeutralObject.h)

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействие юнитов. Основные требования к классам нейтральных объектов поля:

- Создано не менее 4 типов нейтральных объектов

и

- Классы нейтральных объектов должны иметь как минимум один общий интерфейс

Изначально был создан класс-интерфейс нейтральных объектов, в котором содержатся виртуальные методы: `whatYouName()` – метод, для отображения ландшафта на поле специальным символом, данный метод возвращает “имя” ландшафта (`char name`), `copy()` – метод копирования ландшафта, `canStepOnIt()` – метод, который сообщает можно ли вставить на клетку(возвращает `true` или `false`) и `operator +()` – про этот метод будет рассказано в следующих пунктах. От него наследуется класс нейтрального объекта `NeutralObject.h` и уже от этого класса наследуются 4 типа нейтральных объектов:

Аптечка – `MedicalKit.h/MedicalKit.cpp` увеличивает здоровье.

```
#include "MedicalKit.h"

MedicalKit::MedicalKit() {
    name = '+';
    bonus = 20;
}

char MedicalKit::whatYouName() {
    return name;
}

NeutralObjectInterface *MedicalKit::copy() {
    return new MedicalKit(*this);
}

bool MedicalKit::canStepOnIt() {
    return true;
}

void MedicalKit::operator+(int *characteristic) {
    *characteristic = *characteristic+bonus;
}
```

Рисунок 13 – `MedicalKit.cpp`

Зелье силы – `PowerPotion.h/ PowerPotion.cpp` увеличивает атаку.

```

#include "PowerPotion.h"

PowerPotion::PowerPotion() {
    name = '&';
    bonus = 10;
}

char PowerPotion::whatYouName() {
    return name;
}

NeutralObjectInterface *PowerPotion::copy() {
    return new PowerPotion(*this);
}

bool PowerPotion::canStepOnIt() {
    return true;
}

void PowerPotion::operator+(int *characteristic) {
    *characteristic = *characteristic+bonus;
}

```

Рисунок 14 – PowerPotion.cpp

Щит – Shild.h/ Shild.cpp увеличивает броню.

```

#include "Shild.h"

Shild::Shild() {
    name = '@';
    bonus = 10;
}

char Shild::whatYouName() {
    return name;
}

NeutralObjectInterface *Shild::copy() {
    return new Shild(*this);
}

bool Shild::canStepOnIt() {
    return true;
}

void Shild::operator+(int *characteristic) {
    *characteristic = *characteristic+bonus;
}

```

Рисунок 15 – Shild.cpp

Камень – Stone.h/ Stone.cpp запрещает становиться на клетку поля.

```
#include "Stone.h"

Stone::Stone() {
    name = 'o';
}

char Stone::whatYouName() {
    return name;
}

NeutralObjectInterface *Stone::copy() {
    return new Stone(*this);
}

bool Stone::canStepOnIt() { // false т к на камень нельзя наступать
    return false;
}

void Stone::operator+(int *characteristic) {
}
```

Рисунок 16 – Stone.cpp

- Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций

Взаимодействие юнитов с нейтральными объектами реализовано в виде перегрузки операции +. Вызов перегруженного оператора производится из метода useStrategy(), который располагается в классе Client.h. Демонстрационные примеры взаимодействия юнитов с нейтральными объектами представлены на рисунках 17,18,19,20.

```

-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*| |
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|A,*|,*|,*|,*|
|,*|E,*|,*|,*|,*|,*|+|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|o|K,*|
|,*|,*|,*|,*|,*|,*|,*|,*|H,*|
|,*|,*|,*|,*|,*|,*|,*|,*|~,*|
-----
Health unit 'A': 100
-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*| |
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|E,*|,*|,*|,*|A,*|+|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|o|K,*|
|,*|,*|,*|,*|,*|,*|,*|,*|H,*|
|,*|,*|,*|,*|,*|,*|,*|,*|~,*|
-----
Health unit 'A': 120

```

Рисунок 17 – Пример увеличения здоровья юнита, после передвижения на клетку с аптечкой.

```

-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*| |
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|A,*|,*|,*|,*|
|,*|E,*|,*|,*|,*|,*|+|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|o|K,*|
|,*|,*|,*|,*|,*|,*|,*|,*|H,*|
|,*|,*|,*|,*|,*|,*|,*|,*|~,*|
-----
Damage unit 'E': 70
-----
|1,*|,*|,*|,*|,*|,*|,*|,*|,*| |
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|A,*|,*|,*|,*|
|,*|,*|E,*|,*|,*|,*|+|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|,*|
|,*|,*|,*|,*|,*|,*|,*|,*|o|K,*|
|,*|,*|,*|,*|,*|,*|,*|,*|H,*|
|,*|,*|,*|,*|,*|,*|,*|,*|~,*|
-----
Damage unit 'E': 80

```

Рисунок 18 – Пример увеличения атаки юнита, после передвижения на клетку с зельем силы.

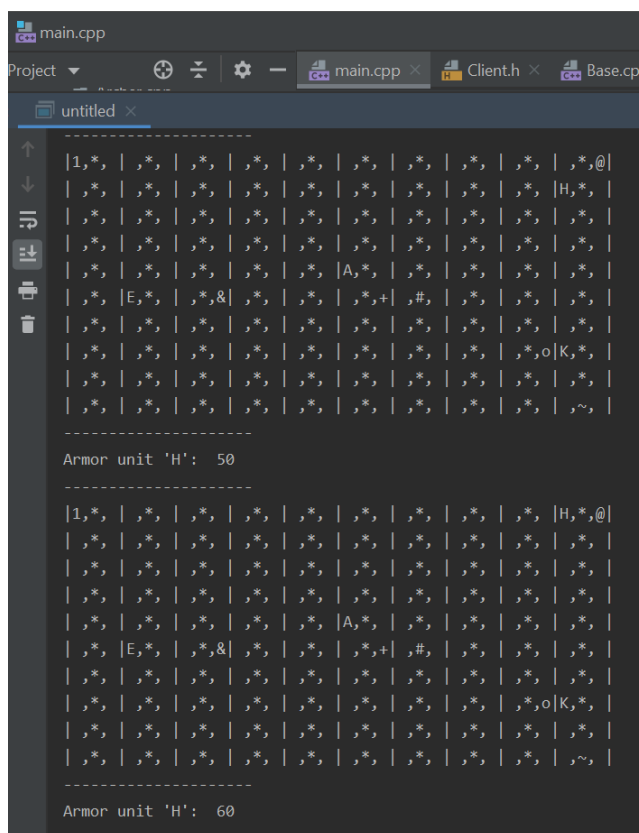


Рисунок 19 – Пример увеличения брани юнита, после передвижения на клетку со щитом.



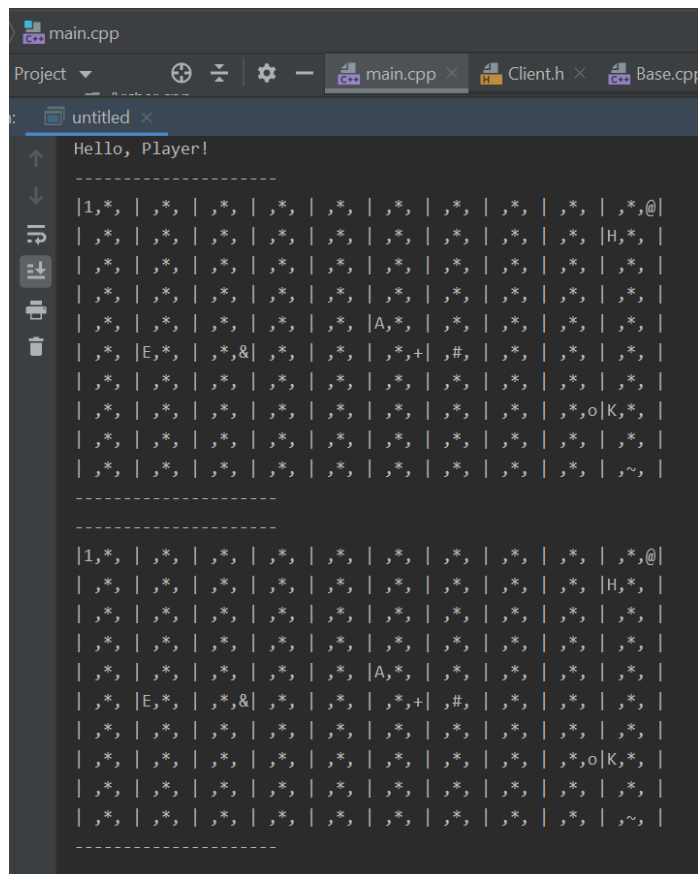


Рисунок 20 – Пример отсутствия передвижения юнита, после команды перейти на клетку с камнем.

Все основные требования выполнены.

### Дополнительные требования

- *\*Для наблюдения над юнитами в классе база используется паттерн “Наблюдатель”*

Был создан паттерн Наблюдатель, для наблюдения над юнитами.

Наблюдатель реализует у класса Base механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов(юнитов) и тем самым наблюдать за ними.

При создании базой юнитов, наблюдатель подписывает базу на юнитов. И при уничтожении юнита, зависящая от наблюдателя база уведомляется и обновляет количество юнитов.

```

class Observer{

public:

    virtual void handleEvent(unsigned numberInArray) = 0;

};

```

Рисунок 21 – Содержание класса Observer

```

void Base::handleEvent(unsigned numberInArray) {
    countUnit--;
    for (unsigned int i = numberInArray; i < controlMax-1; i++) {
        units[i] = units[i+1];
    }
}

```

Рисунок 22 – Реализация метода Наблюдателя.(уменьшается количество юнитов и происходит сдвиг массива юнитов).

```

Archer::~~Archer() {
    observer->handleEvent(numberInArray);
}

```

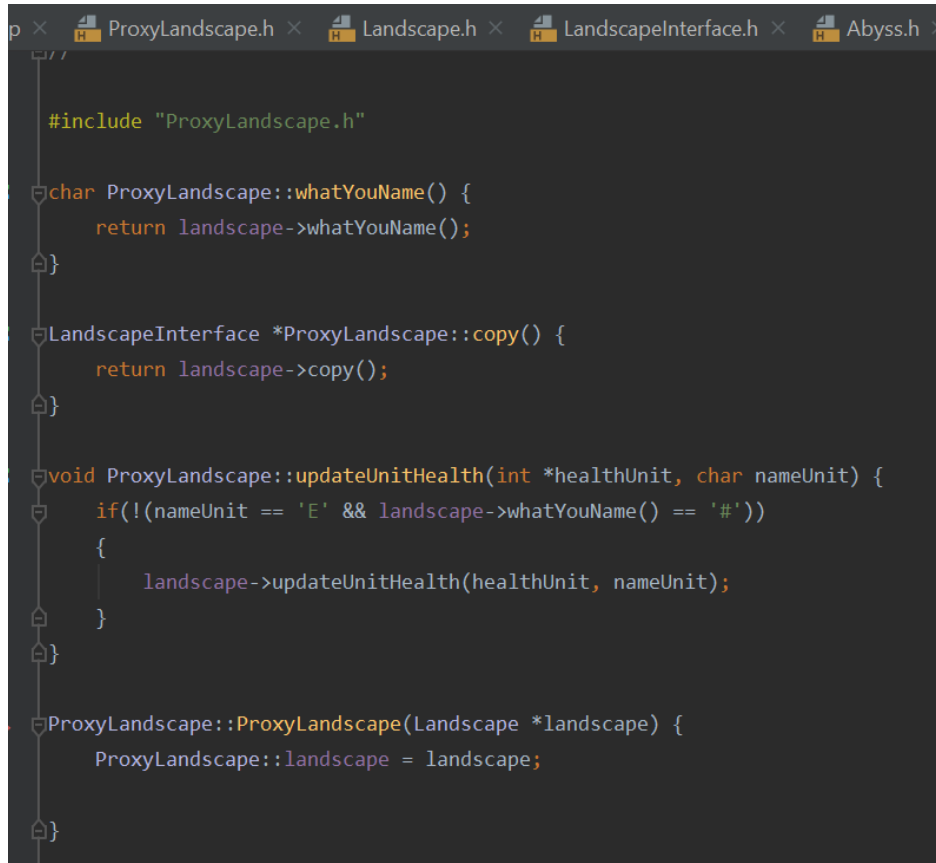
Рисунок 23 – Пример деструктора юнита.

Доказательство работы наблюдателя приведено на рисунке 5, когда база выводит текущее количество юнитов, относящихся к ней.

- *\*Для взаимодействия ландшафта с юнитам используется паттерн "Прокси"*

Был реализован класс ProxyLandscape, который наследуется от класса Landscape. В поле ProxyLandscape хранится указатель на ландшафт(Landscape\* landscape).

В данной реализации игры, юнит ElfOnUnicorn может летать, а следовательно, ландшафт пропасть('#') для него не является угрозой. Поэтому паттерн Прокси учитывает это. Сам метод `updateUnitHealth()` вызывается в методе `move()` класса `Field`.



```
//
#include "ProxyLandscape.h"

char ProxyLandscape::whatYouName() {
    return landscape->whatYouName();
}

LandscapeInterface *ProxyLandscape::copy() {
    return landscape->copy();
}

void ProxyLandscape::updateUnitHealth(int *healthUnit, char nameUnit) {
    if(!(nameUnit == 'E' && landscape->whatYouName() == '#'))
    {
        landscape->updateUnitHealth(healthUnit, nameUnit);
    }
}

ProxyLandscape::ProxyLandscape(Landscape *landscape) {
    ProxyLandscape::landscape = landscape;
}
```

Рисунок 24 – Содержание файла `ProxyLandscape.cpp`

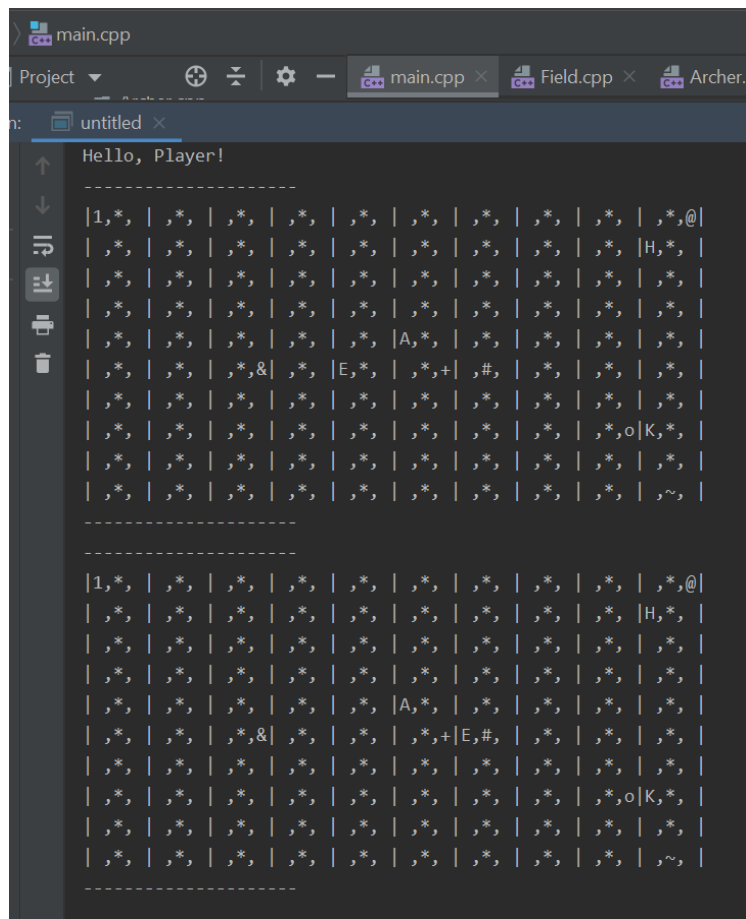
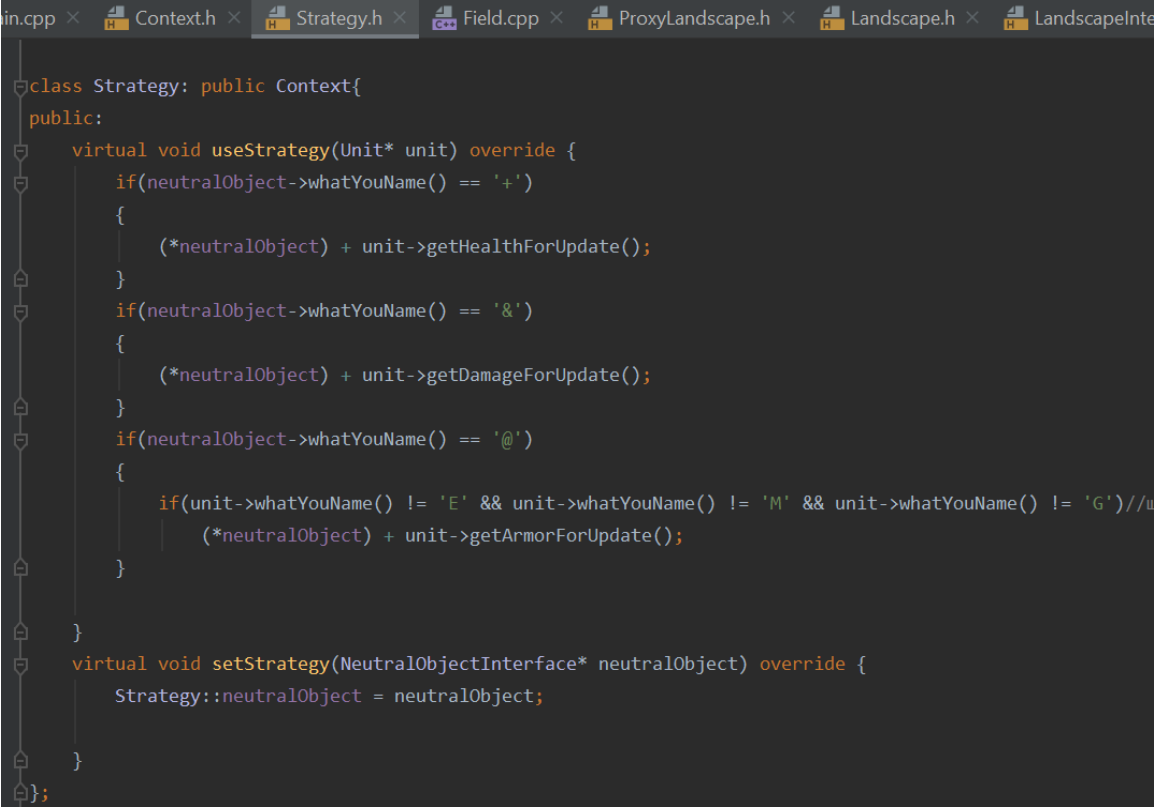


Рисунок 25 – Демонстрационный пример передвижения юнита ElfOnUnicorn на клетку поля с ландшафтом пропасть(юнит не убит).

- *\*Для взаимодействия одного типа нейтрального объекта с разными типами юнитов используется паттерн “Стратегия”*

Было создано 2 файла: Context.h и Strategy.h. Задумка игры заключается в том, что магические персонажи(эльф, маг и гном) не могут пользоваться нейтральным объектом щит. Щит – это предмет сильных героев. Следовательно, по типу юнита определяется, следует применить щит к юниту или нет.



```

class Strategy: public Context{
public:
    virtual void useStrategy(Unit* unit) override {
        if(neutralObject->whatYouName() == '+')
        {
            (*neutralObject) + unit->getHealthForUpdate();
        }
        if(neutralObject->whatYouName() == '&')
        {
            (*neutralObject) + unit->getDamageForUpdate();
        }
        if(neutralObject->whatYouName() == '@')
        {
            if(unit->whatYouName() != 'E' && unit->whatYouName() != 'M' && unit->whatYouName() != 'G')//ш
            {
                (*neutralObject) + unit->getArmorForUpdate();
            }
        }
    }
    virtual void setStrategy(NeutralObjectInterface* neutralObject) override {
        Strategy::neutralObject = neutralObject;
    }
};

```

Рисунок 26 – Содержание Strategy.h

Таким образом, класс Strategy реализует паттерн Стратегия

### Выводы.

В ходе лабораторной работы был разработан и реализован набор интерфейсов классов, получены навыки в работе с ними и перегрузкой операций.