

**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

\_\_\_\_\_

Максимова А.А.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2020

### **Цель работы.**

Разработать и реализовать набор классов: класс базы, набор классов ландшафта карты, набор классов нейтральных объектов поля.

### **Постановка задачи.**

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

### Основные классы, реализованные в программе.

Класс	Назначение	Файлы
Attribute	Абстрактный класс, от которого наследуются классы Damage, Health, Armor - классы атрибутов юнитов.	Attribute.cpp Attribute.h
Base	Абстрактный класс, от которого наследуются классы BaseMonster, BaseHero, BasePeople - базы соответствующих типов юнитов.	Base.h Base.cpp
Cell	Клетка содержит указатели на юнита, ландшафт, нейтральный объект и базу, находящиеся на ней, при отсутствии соответствующего объекта на клетке указатель на этот объект равен nullptr.	Cell.h Cell.cpp
Control	Класс контролирующей очередность ходов игроков, необходим для вызова методов класса Gamer в порядке соответствующем логике игры.	Control.h Control.cpp
Dialog	Класс, содержащий методы для вывода текстовой информации, относящейся к игре (правила, описание персонажей)	Dialog.h Dialog.cpp
GameField	Класс игрового поля предназначен для создания контейнера (двумерный массив типа клетка) и работы с ним (добавление	GameField.h GameField.cpp

	нейтральных объектов, ландшафта, базы, юнитов на поле; перемещение юнитов; удаление объектов с поля).	
Gamer	Класс, методы которого связывают команды, получаемые от игрока, с функционалом игрового поля (также в классе производится обработка ошибок в введенных пользователем данных).	Gamer.h Gamer.cpp
Landscape	Абстрактный класс, от которого наследуются конкретные типы ландшафта игрового поля (Grass, Lake, ritualCircle, Pit).	Landscape.h Landscape.cpp
NeutralObject	Абстрактный класс, от которого наследуются конкретные типы нейтральных объектов, размещаемых на игровом поле (HolyCross, Snake, Tablet, SurpriseBox).	NeutralObject.h NeutralObject.cpp
Unit	Абстрактный класс, от которого наследуются классы разных типов юнитов.	Unit.h Unit.cpp
Object	Абстрактный класс, от которого наследуются классы Unit, Base, NeutralObject.	Object.h Object.cpp
AbstractFactory	Класс, необходимый для создания юнитов.	AbstractFactory.h AbstractFactory.cpp

## Выполнение требований к классу базы.

Для реализация базы был создан абстрактный класс Base, от которого наследуются три конкретных типа баз: база Монстров, база Героев и база Людей (см. рис. 1).

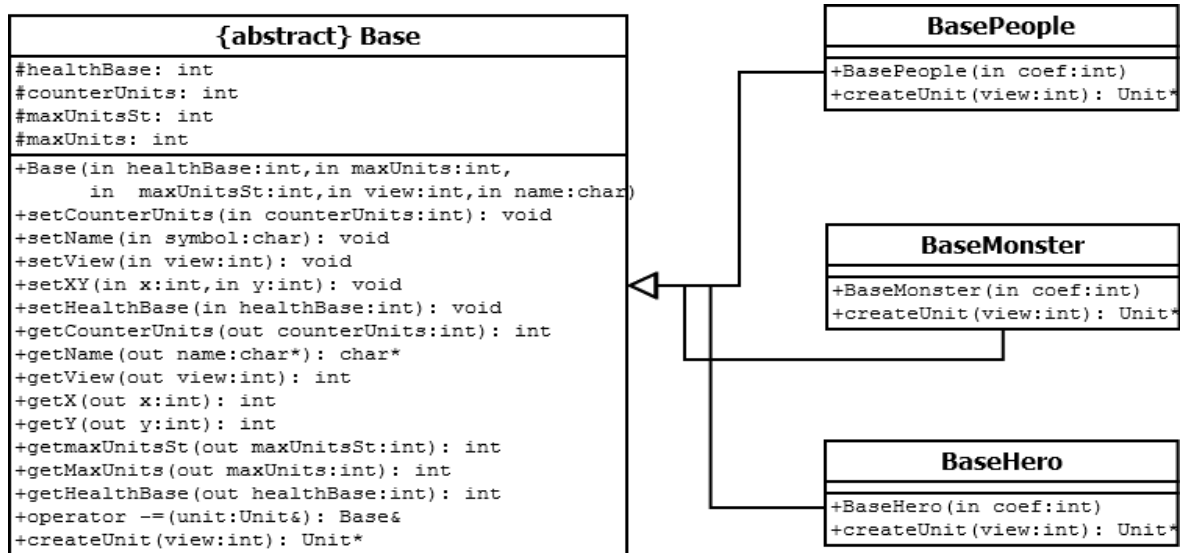


Рисунок 1 - UML диаграмма классов баз

### 1. Размещение базы на поле

Добавление баз на поле происходит в методе `void GameField::addBases()`, где каждая база ставится на определенную позицию: так база Монстров располагается в нижнем левом углу поля, база героев - в правом нижнем углу, и база Людей - по середине поля сверху. При добавлении базы на поле, устанавливаются значения ее координат (x, y). Причем каждый из типов базы может быть создан лишь один раз. Параметр `step` - коэффициент масштабирования, чем больше размер поля, тем больше количество юнитов в каждой команде, и размер одного шага юнита при перемещении.

```
void GameField::addBases() {
    field[0][width / 2].setBase(new BasePeople(step));
    field[0][width / 2].getBase()->setXY(0, width / 2);

    field[height - 1][0].setBase(new BaseMonster(step));
    field[height - 1][0].getBase()->setXY(height - 1, 0);

    field[height - 1][width-1].setBase(new BaseHero(step));
    field[height - 1][width - 1].getBase()->setXY(height - 1, width - 1);
}
```

Рисунок 2 - Добавление баз на поле

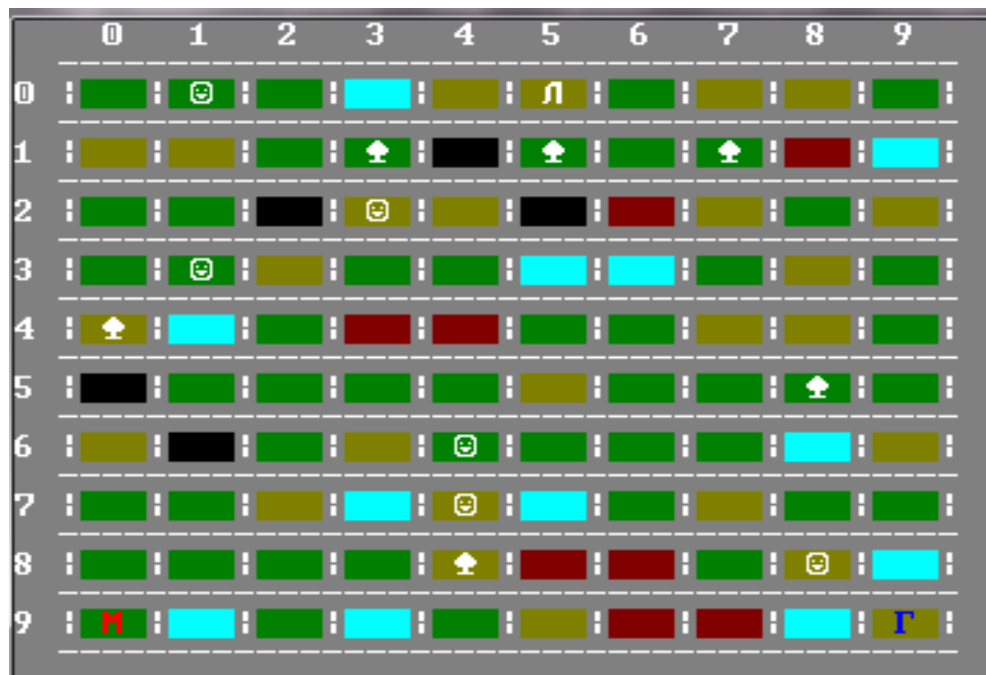


Рисунок 3 - Расстановка баз на поле (М, Л, Г)

## 2. Методы создания юнитов

Каждая конкретная база имеет метод `createUnit`, продемонстрированный на рис. 4, принимающий параметр `view` - вид юнита, который должен быть создан, и возвращающий указатель на созданного юнита. Метод обращается к конкретным фабрикам, наследуемым от класса Абстрактная фабрика. Юниты создаются только базой, поэтому при "разгроме" базы, добавление новых юнитов на поле становится невозможным.

```
class BasePeople : public Base {
public:
    BasePeople(int coef);

    Unit* createUnit(int view) override {
        ConcreteFactoryPeople people;
        return people.createUnit(view);
    }
};
```

Рисунок 4 - метод `createUnit` в классе Базы Людей

### 3. Учет юнитов и реакция на их уничтожение

Для учета юнитов в классе базы имеется поле counterUnits, хранящие текущее количество юнитов, относящихся к одной базе. В начале игры, это значение у команд Монстров и Героев равно, по правилам игры, а у Людей равно удвоенному значению. Для удобства количество юнитов выводится в консоль и постоянно обновляется (см. рис. 5). При уничтожении юнита, то есть удалении с поля, данный параметр уменьшается, реализация см. на рис. 6.

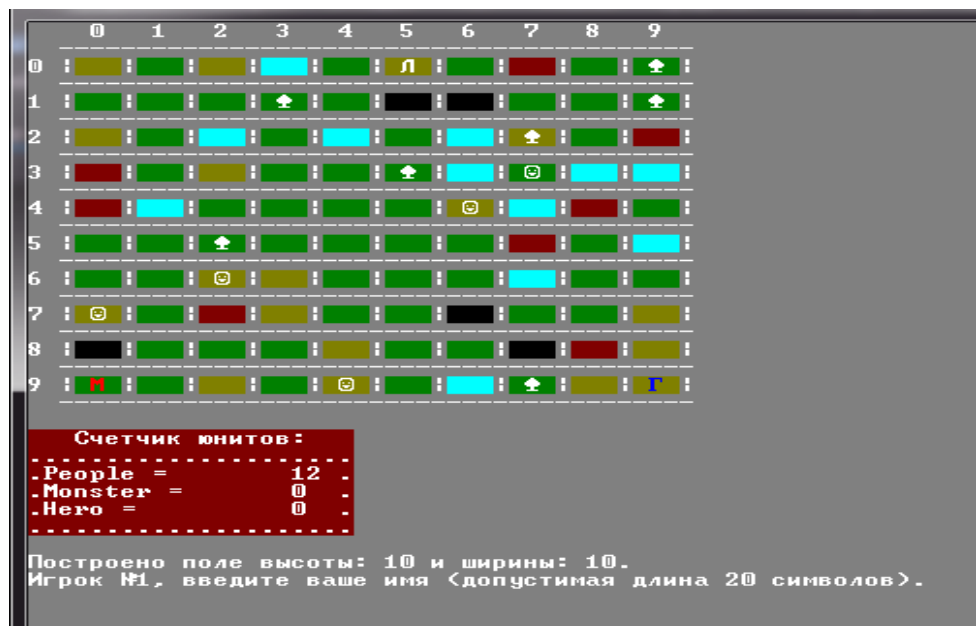


Рисунок 5 - Счетчик юнитов

```
GameField::deleteUnit(int x, int y) {
    if (field[x][y].getUnit() != nullptr) {
        //фиксируем смерть в базе
        if (field[x][y].getUnit()->getView() == 1 || field[x][y].getUnit()->getView() == 2) //monster
            field[height - 1][0].getBase()->setCounterUnits(field[height - 1][0].getBase()->getCounterUnits() - 1);
        if (field[x][y].getUnit()->getView() == 3 || field[x][y].getUnit()->getView() == 4) //hero
            field[height - 1][width - 1].getBase()->setCounterUnits(field[height - 1][width - 1].getBase()->getCounterUnits() - 1);
        if (field[x][y].getUnit()->getView() == 5 || field[x][y].getUnit()->getView() == 6) //people
            field[0][width / 2].getBase()->setCounterUnits(field[0][width / 2].getBase()->getCounterUnits() - 1);

        delete field[x][y].getUnit();
        field[x][y].setUnit(nullptr);
        return 0;
    }
    else {
        return 1;
    }
}
```

Рисунок 6 - Уничтожение юнита

#### 4. Характеристики базы

```
class Base : public Object {  
protected:  
    int healthBase;  
    int counterUnits;  
    int maxUnitsSt;  
    int maxUnits;
```

Рисунок 7 - Поля класса Base

healthBase - целочисленная переменная, хранящая значение здоровья базы, в игре = это значение варьируется от 10 до 15 тысяч, что примерно в 10-15 раз, больше значения урона наносимого любым юнитом - то есть для уничтожения вражеской базы необходимо произвести большое количество атак.

База используется для создания юнитов как в начале игры, так и во время.

В начале пользователь сам решает юнита какого вида создать и где его расположить (в пределах своей половины поля), поэтому количество юнитов, ограничивается фиксированной переменной maxUnitsSt - то есть максимальное количество юнитов в каждой команде.

Во время игры, пользователь не может самостоятельно добавить юнита на поле. Юниты, если в этом есть необходимость, добавляются автоматически: через каждый два полных хода (то есть каждый игрок сходил два раза) добавляются монстры и герои, через каждые три хода - люди.

Координаты расположения юнитов и конкретный вид в таком случае определяются рандомно. Количество юнитов ограничивается переменной maxUnits - сколько максимально юнитов может быть создано одновременно (автоматически) на базе, при этом максимальное количество юнитов, относящихся к одной базе, также ограничивается числом maxUnitsSt.

Например, если maxUnitsSt = 12, maxUnits = 4, а текущее количество юнитов на поле counterUnits = 3, то автоматически будет добавлено 4 юнита. Если counterUnits = 11, то автоматически будет добавлен только 1 юнит.



## Выполнение требований к набору классов ландшафта.

1. Все классы ландшафта должны иметь как минимум один интерфейс.

Интерфейс классов ландшафта (Grass, Lake, Pit, ritualCircle) реализуется посредством наследования от абстрактного класса Landscape (см. рис. 9), в котором определены поля color - номер цвета ландшафта для вывода при печати поля, type - целочисленное значение типа ландшафта, необходимое при работе с ландшафтом в других классах; геттеры, а также две виртуальные функции - одна (getNewColor) для определения значения поля color в зависимости от объекта, находящегося на ландшафте, вторая для реализации влияния ландшафта на юнитов (пункт 2).

```
class Landscape { //абстрактный класс
protected:
    int color;
    int type;
public:
    int getColor();
    int getType();
    virtual int noPassage(int viewUnit) = 0;
    virtual int getNewColor(int viewObject) = 0;
};
```

Рисунок 8 - Абстрактный класс

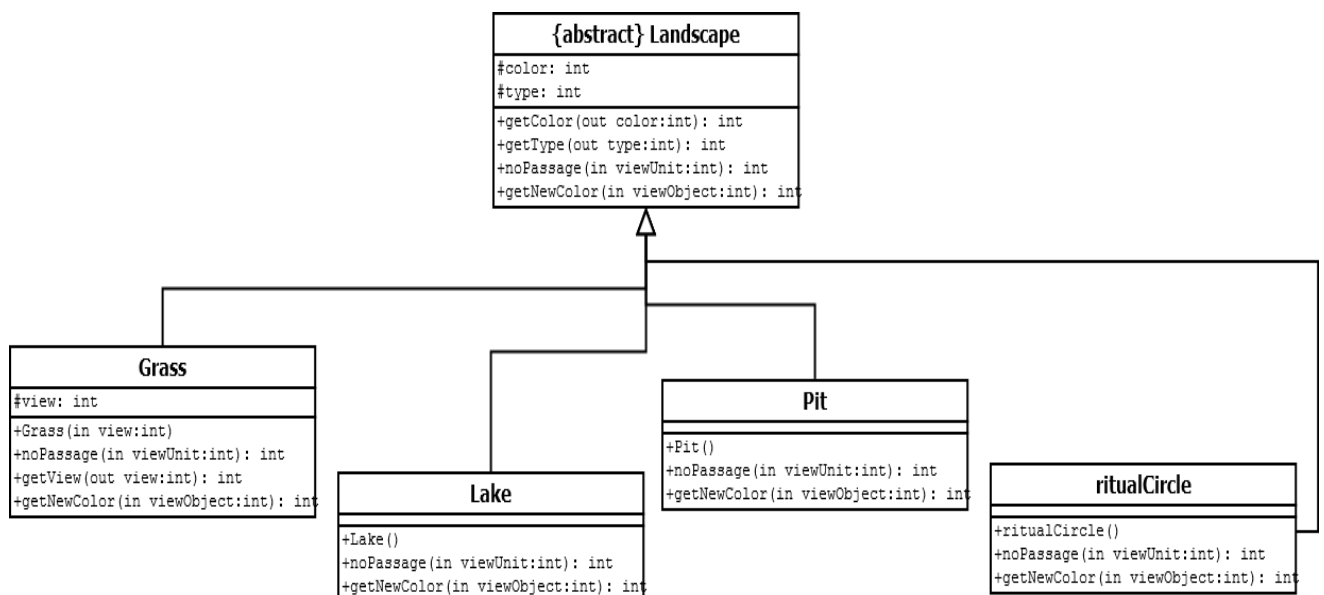


Рисунок 9 - UML диаграмма классов ландшафта

### *2. Ландшафт должен влиять на юнитов.*

Согласно таблице 1, в каждом конкретном классе ландшафта переопределяется метод `noPassage`, принимающий вид юнита, и возвращающий 1, в случае запрета прохода.

Таким образом, при вызове функции перемещения юнита, проверяется на какой тип ландшафта он хочет перейти.

Базы юнитов ставятся только на `Grass`, из тех соображений, что каждый юнит должен иметь доступ к своей базе.

Нейтральный объекты ставятся на любой тип ландшафта, за исключением `Pit`, так как в таком случае, невозможно будет получить данный объект.

Класс `Grass` имеет поле `view`, необходимое только с точки зрения графики, так как трава занимает большую часть ландшафты, то она, в зависимости от значения `view`, имеет разные оттенки окраски. Пример покрытия поля ландшафтом продемонстрирован на рис. 3.

Табл. 1 - Влияние ландшафтов на юнитов: проходимость

Landscape	Monster		Hero		People	
	Vampire	Werewolf	Healer	Magician	Man	Hunter
Grass	+	+	+	+	+	+
Lake	-	-	+	-	-	-
Ritual Circle	-	-	-	+	-	-
Pit	-	-	-	-	-	-

### *3. На каждой клетке поля должен быть определенный тип ландшафта.*

Все игровое поле сразу после создания покрывается ландшафтом - реализуется в методе `GameField::coveredLands()`, вычисляется общая площадь игрового поля `area`, и в зависимости от этого значения, рассчитывается количество клеток, покрытое определенным ландшафтом - так `Grass`, проходимый для любого юнита, покрывает наибольшую часть поля, а `Pit` -

наименьшую. При этом расположение конкретного типа ландшафта на поле определяется случайно.

```
GameField::coveredLands() { //покрытие ландшафтом
    srand(time(NULL));
    int x, y, view;
    int area = width * height;
    int countGrass = (area * 3) / 4; //75%
    int countLake = (area - countGrass) / 2; //13%
    int countCircle = ((area - (countGrass + countLake)) * 2) / 3; //8-10%
    int countPit = area - (countGrass + countLake + countCircle);
}
```

Рисунок 10 - Распределение ландшафта по полю

### Выполнение требований к набору классов нейтральных объектов.

1. Классы нейтральных объектов должны иметь минимум один общий интерфейс

Все классы конкретных нейтральных объектов наследуются от абстрактного класса NeutralObject (см. рис. 11), в котором поле name - символ, для обозначения объекта на поле при печати, healthEffect - целочисленное значение взаимодействия на здоровье юнита, damageEffect - на урон, также определены геттеры и сеттеры для координат расположения x, y; имени и вида, а также объявлена виртуальная функция перегрузки операции +=.

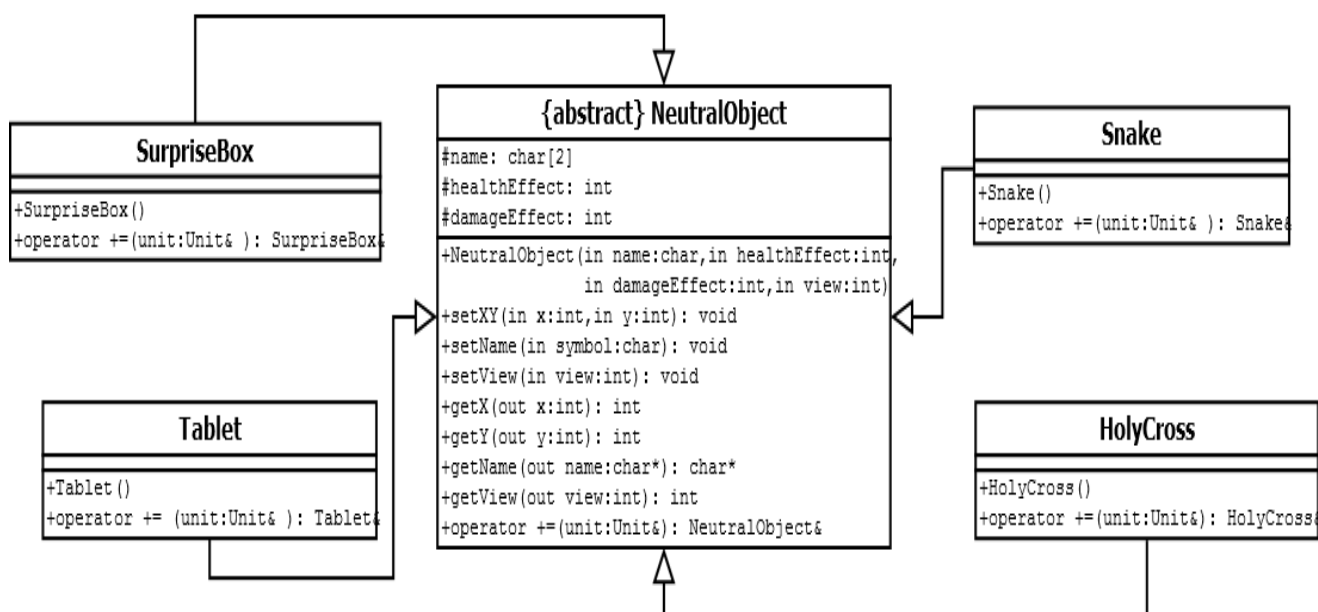


Рисунок 11 - UML диаграмма классов нейтральных объектов

## 2. Создано не менее 4 типов нейтральных объектов

Табл. 2 - Типы нейтральных объектов

Тип нейтрального объекта	Действует	Эффект
<u>HolyCross</u>	На Вампиров и Магов	-100 здоровья
<u>Snake</u>	На Людей, Охотников, Магов, Целителей	-100 здоровья
<u>Tablet</u>	На всех	+150 здоровья
<u>SurprizeBox</u>	На всех	+200 к здоровью или +200 к урону

## 3. Взаимодействие юнитов с нейтральными объектами, реализованное в виде перегрузки операций

Для реализации взаимодействия юнитов с нейтральными объектами используется перегрузка оператора += (для пользовательских классов NeutralObject и Unit), объявленная как виртуальная функция и переопределяемая в каждом классе в соответствии с табл. 2.

Вызов перегруженного оператора производится из метода `int GameField::movingUnit(int x, int y, int x2, int y2, int steps)`, где после взаимодействия вызывается удаление нейтрального объекта, и проверяется состояние юнита (если здоровье  $\leq 0$ ), то это означает, что юнит умер и также отправляется в метод удаления.

### **Взаимодействие юнитов (через перегрузку операторов).**

Кроме взаимодействия юнитов с нейтральными объектами, также было реализовано их взаимодействие друг с другом, и взаимодействие юнитов с базой.

Для взаимодействия юнитов друг с другом был перегружен оператор `-=`, в методе класса `Unit`, левым и правым операндом являются юниты.

Сначала, при нападении юнита, уменьшается значение брони, если броня на исходе, уменьшается здоровье. Перегруженный оператор также вызывается в методе игрового поля `movingUnit(int x, int y, int x2, int y2, int steps)`, где проверяется здоровье юнита (если меньше или равно 0, то юнит умер), нападающий и потерпевший юниты меняются местами.

```
Unit& Unit::operator -= (Unit& unit2) { //атака
    if (unit2.getArmor() >= this->getDamage()) {
        unit2.setArmor(unit2.getArmor() - (this->getDamage())); //броня
    }
    else {
        unit2.setHealth(unit2.getHealth() - (this->getDamage())); //здоровье
    }
    return *this;
}
```

Рисунок 12 - Перегрузка оператора `-=`

Для взаимодействия юнита с базой, реализована перегрузка оператора `-=`, в классе базы. Вызов перегруженного оператора снова вызывается из метода игрового поля для перемещения, где проверяется состояние здоровья базы. При этом если база остается целой, то юнит не перемещается на ее позицию. То есть атака на данный момент реализуется как передвижение на тот объект, который должен быть атакован.

Примеры атаки см. в демонстрационных примерах.

#### **Ход действий в игре:**

1. Игроки имеют возможность ознакомиться с правилами игры.
2. Вводится размер игрового поля.
3. Создается контейнер, поле заполняется ландшафтом, люди рандомно расставляются по полю.
4. Далее каждый из игроков ставит своих юнитов на поле, имея возможность в случае ошибки переставить их на другие позиции.

5. После начинается игра: за один ход можно перемещать своих юнитов 5 раз, нападая на других юнитов и базы, собирая нейтральные объекты.

6. Игра продолжается до тех пор, пока количество людей не стало равно нулю.

Скорее всего действия и правила игры будут еще изменяться.

### Дополнительно.

Был прописан класс Cell - клетка, содержащий указатели на юнита, ландшафт, нейтральный объект и базу, сеттеры и геттеры для них.

```
class Cell {
private:
    Unit* unit;
    Landscape* landscape;
    NeutralObject* neutralObject;
    Base* base;

public:
    Cell();
    void setUnit(Unit* unit);
    void setLand(Landscape* landscape);
    void setNeutralObject(NeutralObject* neutralObject);
    void setBase(Base* base);

    Unit* getUnit();
    Landscape* getLand();
    NeutralObject* getNeutralObject();
    Base* getBase();
};
```

Рисунок 13 - класс Cell

Игровое поле, которое ранее было объявлено как двумерный массив указателей на объекты, теперь является двумерным массивом из клеток.

Метод печати игрового поля был перенес в класс Gamer.

Были добавлены проверки на корректность ввода данных пользователем - исключение ошибок, например, появился метод DataCorrect, обрабатывающий ошибки типа "ожидается число, введен символ".

## Демонстрационные примеры.

### 1. Обработка ошибок:

```
Добро пожаловать в игру "Mystic Strategy"
Если вы хотите ознакомиться с правилами игры, введите "y".
Если вы уже знаете правила и хотите их пропустить – введите "n".
ereur
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
34
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
```

Рисунок 13 - Неверный ввод символа

```
Перед началом игры введите высоту и ширину игрового поля.
yrrere
Ошибка. Данные некорректны! Ожидается число.
2 2
Ошибка: неверно указан размер поля!
Создано поле стандартного размера.
```

Рисунок 14 - Некорректный размер игрового поля

```
Построено поле высоты: 10 и ширины: 10.
Игрок #1, введите ваше имя (допустимая длина 20 символов).
kolua
kolua, выберите за какую команду вы будете играть.
Если вы хотите играть за монстров введите "m".
Если вы хотите играть за героев введите "h".
ereur
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
34
Ошибка: введен некорректный символ. Попробуйте еще раз.
Ошибка: введен некорректный символ. Попробуйте еще раз.
```

Рисунок 15 - Неверный ввод символа

```
kolua, вам необходимо добавить 6 персонажей на поле.
Для этого вам необходимо ввести три числа:
Первое: номер команды добавления
ВАМПИР: команда для добавления "1".
ОБОРОТЕНЬ: команда для добавления "2".
Два числа – координаты расположения героя (x; y).
Ваше поле левое.
6 2 3
Ошибка ведена неверная команда?
Введите снова:
1 2 0
Позиция занята!
Персонаж не создан! Введите снова!
1 2 2
Ошибка: вы пытаетесь разместить юнитов на поле соперника!
Введите снова:
```

Рисунок 16 - Некорректные команды

## 2. Взаимодействие с нейтральными объектами:

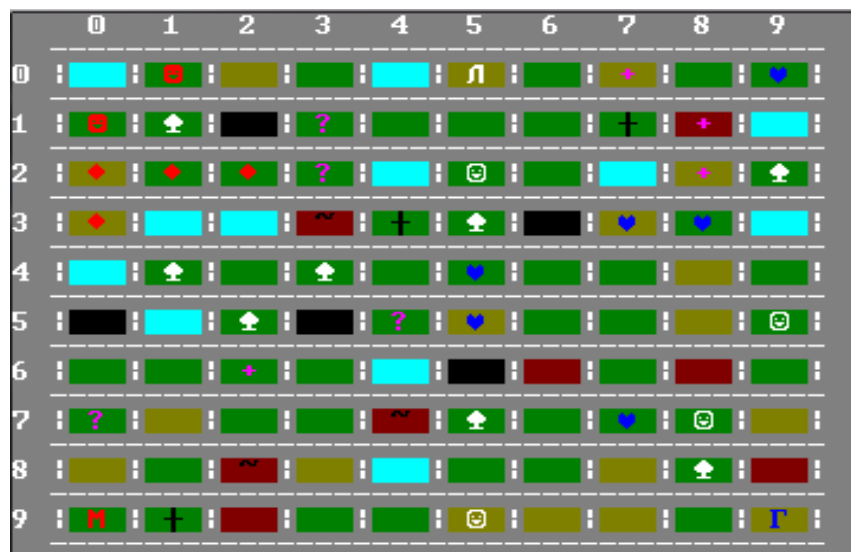


Рисунок 17 - Начальное состояние



Рисунок 18 - Передвижение с клетки (2; 2) на клетку (2; 3), здоровье +200



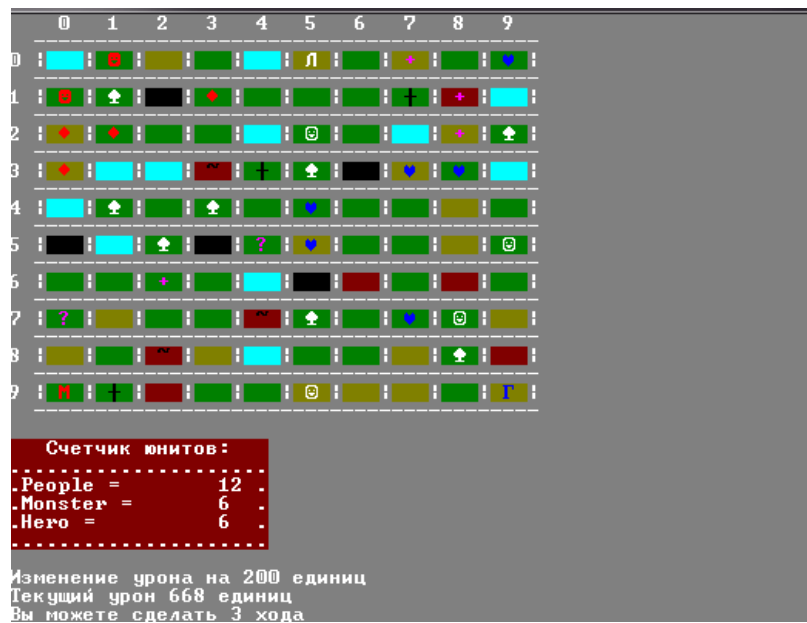


Рисунок 19 - Перемещение с позиции (2; 3) на позицию (1; 3), урон +200

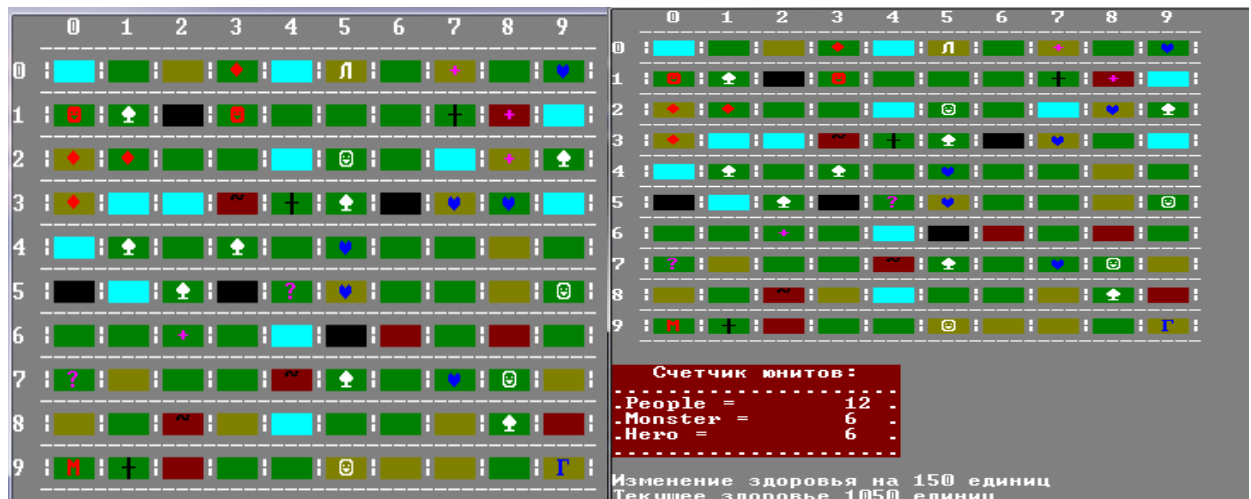


Рисунок 20 - Перемещение с позиции (3; 8) на позицию (2; 8), здоровье +150

### 3. Взаимодействие юнитов друг с другом:

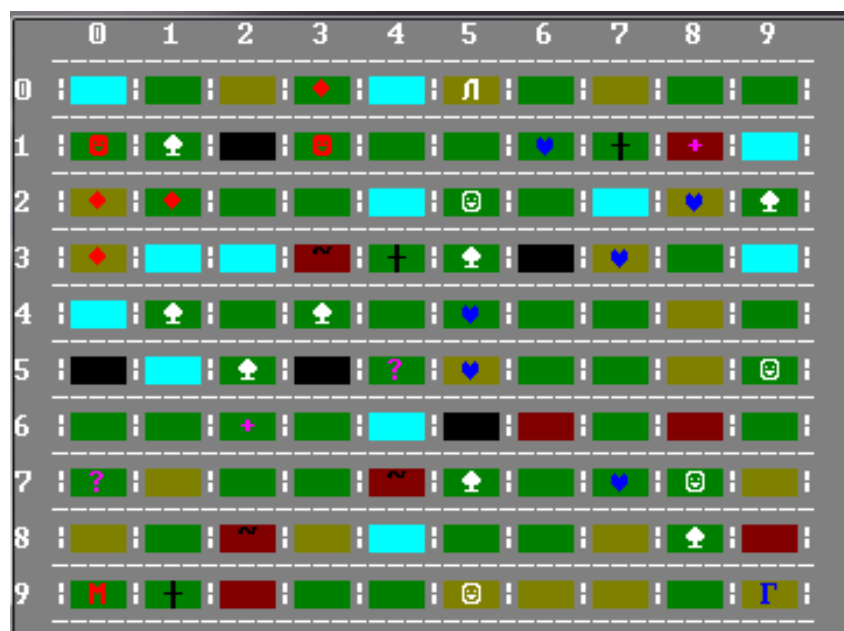


Рисунок 21 - Начальное положение

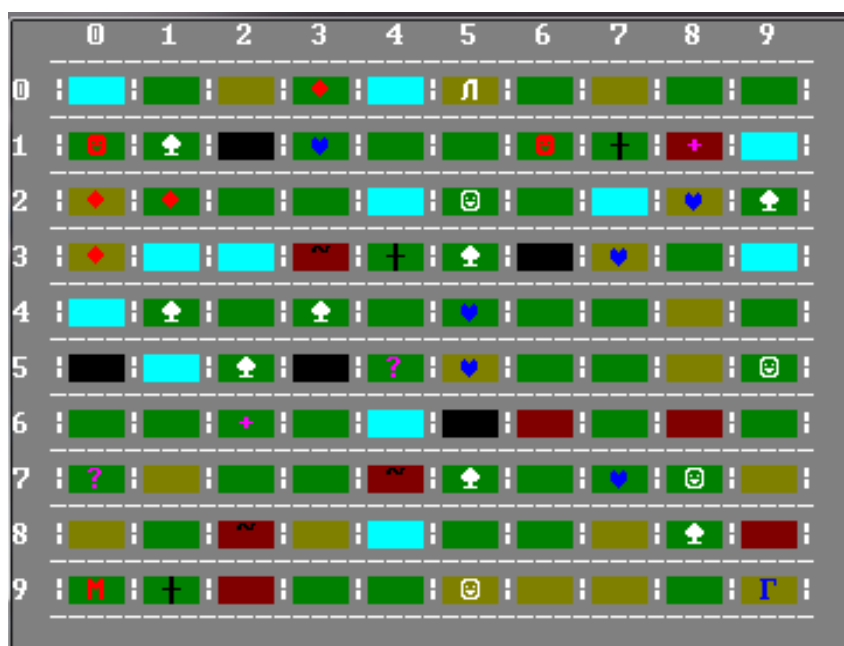


Рисунок 22 - Нападение юнита с позиции (1; 3) на юнита на позиции (1; 6) -  
после нападения поменялись местами

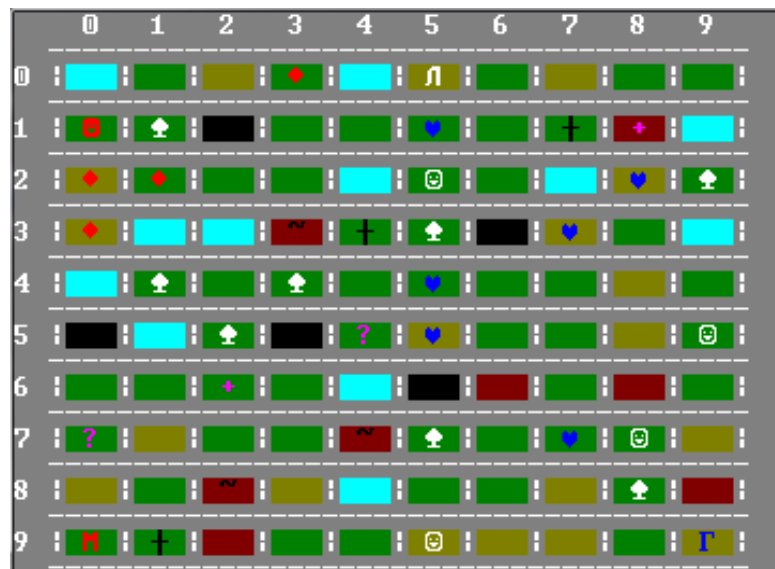


Рисунок 23 - Нападение юнита с позиции (1; 5) на юнита на позиции (2; 5)

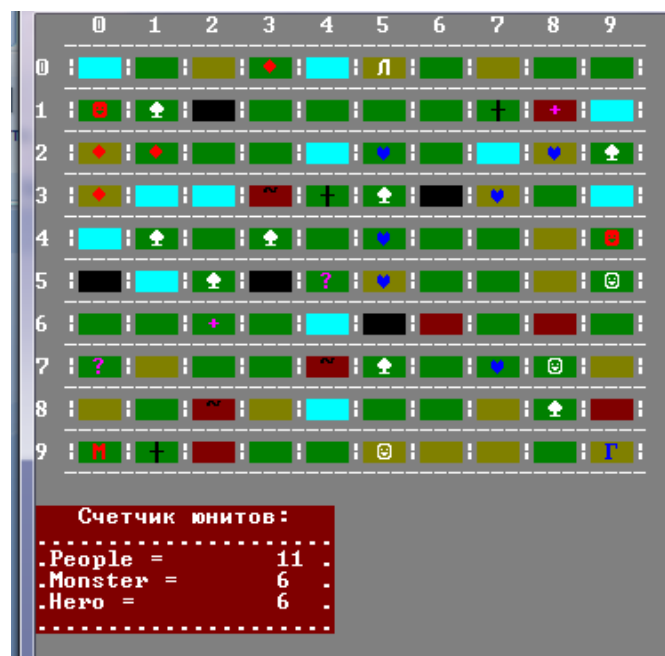






Рисунок 24 - Результат нападения

#### 4. Подсказка для пользователя:

Обозначения на игровом поле:	
ЮНИТЫ:	
ВАМПИР:	☠ тип монстр
ОБОРОТЕНЬ:	♦ тип монстр
ЦЕЛИТЕЛЬ:	♥ тип герой
МАГ:	✳ тип герой
ЧЕЛОВЕК:	☺ тип человек
ОХОТНИК:	♆ тип человек
НЕЙТРАЛЬНЫЕ ОБЪЕКТЫ:	
СВЯТОЙ КРЕСТ:	✝ Св-во: - здоровье у вампиров и магов
ЗМЕЯ:	~ Св-во: - здоровье у всех, кроме вампиров и магов
ТАБЛЕТКА:	+ Св-во: + здоровье любому юниту
КОРОБКА-СЮРПРИЗ:	? Св-во: + здоровье или урон любому юниту
ЛАНДШАФТ:	
	- трава. Может наступать любой юнит.
	- озеро. Нельзя наступать никому кроме целителя.
	- ритуальный круг. Можно наступать только магам.
	- яма. Никому нельзя наступать.
БАЗЫ:	
Л	- база людей
М	- база монстров
Г	- база героев

#### Выводы.

Разработаны и реализованы: наборы классов базы, ландшафта и нейтральных объектов.

