**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Консольная игра»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6382 |  | Черкасова Е.И. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2017

**Цель работы.**

Реализовать класс Объект, а также класс Поле\_боя, являющийся обёрткой над контейнерами-армиями, хранящим экземпляры класса Объект.

**Постановка задачи.**

Объект должен иметь:

1. Двумерные координаты.
2. Очки жизней.
3. Говорящий конструктор, говорящий деструктор.
4. Функцию, которая принимает урон.
5. Функцию, отвечающую на вопрос, есть ли Объект на указанной позиции.
6. Считываться из файла.

Поле\_боя должно иметь:

1. Контейнеры, хранящие объекты.
2. Отрисовку Поля\_боя на экране (разными цветами).
3. Загружать объекты из файла
4. Определять, располагается ли на заданных координатах объект той или иной армии.

**Дополнения:**

**1.** Обновить класс Объект таким образом, чтобы в нём вёлся учёт созданных объектов. Так, чтобы каждому новому создаваемому объекту присваивался новый идентификатор. Добавить “корону” - индикатор, что ещё существует хотя бы один объект. Добавить перегрузку операторов << и >> для Объекта и для Поля боя. Оформить циклы for в с++11 стиле (с использованием auto).

**2.** Создать класс-наследник от класса Объект класс Воин и класс Здание.

Класс Воин должен иметь:

1. Поле, указывающее силу удара.

2. Функцию перемещения вверх/вниз/вправо/влево.

**3.** Функцию атаки объектов, которые располагаются на тех же координатах, что и

он сам.

Класс здание должен иметь:

1. Набор координат, в которых это здание располагается.

**4.** Создать классы-наследники от класса Воин класс Мечник и класс Маг.

Класс Воин должен иметь:

1. Функцию, переопределяющую функцию атаки так, чтобы был атакован объект с наименьшим количеством жизней на соседней клетке.

Класс Маг должен иметь1. Функцию, переопределяющую функцию атаки так, чтобы был атакован объект с наименьшим количеством жизней, а также все прочие объекты на линии атаки (луч атаки имеет ограниченную дальность).

Класс Здание необходимо обновить, сделав его шаблонным, показывая, войнов какого типа это здание производит

Реализовать пошаговую игру до уничтожения одной командой другой.

**Реализация.**

Для решения задачи реализованы классы Object, Field, Crown, а также производные от Object: классы Warrior и Building. Ввод данных осуществляется из файла (имя файла input). Имееются говорящие конструктор и деструктор в обоих классах.

Объекты хранятся в контейнере list (собственный контейнер). Учет объектов ведется с помочью статической переменной (счетчика), находящейся в поле класса Object (count\_ob) и записывается в поле id при создании очередного объекта как его идинтификатор. Доступ к объектам осуществляется по индексам с указанием метки ('o' - объект или здание; 'w' - воин) или id.

Для хранения зданий используется вектор, состоящий из пар элементов (x, y). Воины могут передвигаться по полю (есть проверка границ поля), при совпадении собственных координат с координами какого-либо объекта или здания воин может нанести урон, равнй силе своего удара. Если цель поражена, это отображается на карте. До удара проверяется положение цели, цвет армии воина и цели, при не совпадение атака не будет совершена.

Атака совершается через функцию из класса Field attack(Object\* o), принимающая на вход указатель на воина, совершающего атаку. Вызывает атаку (attack\_aim) у переданного объекта (срабатывает соответсвующая перегрузка в зависимости от типа воина), возвращается вектор из пар координат для атаки. Если воин - мечник, вызывается функция атаки минимального объекта на соседней клетке (weakest\_attack из класса Field), которая аналогично находит цель и передает ее на уничтожение в attack.

Когда все объекты уничтожены, корона тоже уничтожается, что ведёт к поражению в игре. Корона храниться как shared\_ptr в членах данных объектов и задается сразу при создании объекта (const shared \_ptr<>).

Для циклов foreach (for c++11) реализован итератор (begin, end), сделаны соответсвующие перегрузки операторов (!=, =) .

На любом этапе программы при неправильном вводе есть возможность исправить вводимые данные и продолжить или же выйти из программы.

**Класс Object:**

**class Object**

{

**char** mark;

**int** x;

**int** y;

**int** hp;

**const** **int** id;

**const** shared\_ptr<Crown> crown;

**static** **int** count\_ob;

**static** **int** cur\_amount;

**public:**

Object(shared\_ptr<Crown> crown, istream &in);

Object(shared\_ptr<Crown> crown, **char** mark, **int** x, **int** y, **int** hp);

Object(Object **const** &ob);

Object(Object&& ob);

~Object();

**void** read\_obj(fstream &f, **int** id);

**void** damage(**int** damage);

**bool** alive();

**int** which\_army();

**bool** check\_crown();

**void** print\_ob();

**void** print\_ob\_simple();

**virtual** **char** draw();

**char** &getmark();

**int** &getx();

**int** &gety();

**int** &gethp();

**const** **int** getid();

**int** &get\_cur\_amount();

**const** **int** &getid() **const**;

**const** shared\_ptr<Crown> getcrown();

Object &operator=(Object&& ob);

Object &operator=(Object &ob);

**bool** operator==(Object **const** &ob);

**friend** istream &operator>>(istream &in, Object &ob);

**friend** ostream &operator<<(ostream &out, Object &ob);

**virtual** **int**& getpower(){}

**virtual** **int**& getrange(){}

**virtual** vector<pair<**int**, **int**>> &getlocation() {}

**virtual** **int** move\_to(**char** action, **int** border\_x, **int** border\_y){}

**virtual** vector<pair<**int**, **int**>> aim\_attack(){}

**virtual** vector<pair<**int**, **int**>> aim\_attack(**int** direction){}

**virtual** Object\* production(**int** x, **int** y){}

};

---------------------------------------------------------------------

**Класс Field:**

**class** Field

{

**int** x\_size;

**int** y\_size;

weak\_ptr<Crown> crown\_red;

weak\_ptr<Crown> crown\_green;

List<Object \*> list;

**public:**

Field() {}

~Field();

**void** init();

**void** print\_field(ostream &out);

**void** print\_list();

**void** read\_file(istream &f);

Object \*get\_obj(**int** x, **int** y, **char** mark);

Object \*get\_obj(**int** id\_found);

**bool** is\_obj(**int** x, **int** y);

**int** find\_index(**int** x, **int** y, **char** mark);

**void** read\_from\_file(istream &in);

**int** attack(**int** x, **int** y, **int** power);

List<Object \*> &getlist();

**int** &get\_x\_size();

**int** &get\_y\_size();

**friend** istream &operator>>(istream &in, Field &fd);

**friend** ostream &operator<<(ostream &out, **const** Field &fd);

};

----------------------------------------------------------------------

**Класс Crown:**

**class** Crown{

**const** string color;

**public:**

Crown(string color);

Crown();

~Crown();

**const** string &getcolor() **const**;

};

**Класс Building:**

**class** Building : **public** Object{

vector<pair<**int**, **int**>> location;

**public:**

Building(shared\_ptr<Crown> crown, **char** &mark, **int** &x, **int** &y, **int** &hp, vector<pair<**int**, **int**>> &location);

~Building();

**char** draw() **override**;

vector<pair<**int**, **int**>> &getlocation() **override**;

**friend** ostream &operator<<(ostream &out, Building &bd);

};

--------------------------------------------------------------------

**Класс Warrior:**

**class** Warrior : **public** Object{

**int** power;

**public:**

Warrior(shared\_ptr<Crown> crown, istream &in);

~Warrior();

**int** move\_to(**char** action, **int** border\_x, **int** border\_y) **override**;

**int** &getpower() **override**;

**char** draw() **override**;

**friend** istream &operator>>(istream &in, Warrior &wr);

**friend** ostream &operator<<(ostream &out, Warrior &wr);

};

**--------------------------------------------------------------------**

**Вывод.**

Создана имитация консольной игры, использовался собственный шаблонный контейнер с внешним итератором, перегружены не обходимые функции. Реализованы необходимые требования к объектам. Созданы новые объекты: воины и здания, а также интерфейс взаимодействия с ними. Воины могут изменять свое положение на поле. Объекты и здания могут получать урон от воинов. В конце концов воины тоже научились убивать друг друга.

В последней реализации появились новые типы воинов: маг и мечник, способные поражать цели под определенным углом по лучу (маг) и объект с наименьшим количеством жизней на соседней клетке (мечник).