**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

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

Тема: **Создание классов, конструкторов классов, методов классов; наследование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8303 |  | Хохлов Г.О. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2020

**Цель работы.**

Разработать и реализовать набор классов, получить навыки в создании классов, конструкторов классов, методов классов и наследовании.

**Ход работы.**

Разработаны и реализованы набор классов:

Класс игрового поля

Набор классов юнитов

Игровое поле реализовано как контейнер для объектов, представляющий прямоугольную сетку. Создано поле произвольного размера. Также осуществляется контроль максимального количества объектов на поле. В коде предусмотрена возможность добавления и удаления объектов на поле и возможность копирования поля (включая объекты на нем).

Все юниты имеют один общий интерфейс. Реализованы 3 типа юнитов (Викинг, лучник, Маг). Реализованы 2 вида юнитов для каждого типа.

Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака. Юнит имеет возможность перемещаться по карте.

**Выводы.**

В ходе лабораторной работы были разработаны и реализованы класс игрового поля и набор классов юнитов. Получены навыки в создании классов, конструкторов классов, методов классов и наследовании.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

#ifndef OOP\_LAB1\_UNIT\_H

#define OOP\_LAB1\_UNIT\_H

#include "Armor.h"

#include "Weapon.h"

#include "Point.h"

class Unit {

friend class GameField;

protected:

Point position;

Armor armor;

Weapon weapon;

int health;

bool isOnField = false;

public:

Unit(Unit \*other);

Unit() {}

Point& getPosition() { return position; }

};

#endif //OOP\_LAB1\_UNIT\_H

#include <iostream>

#include "GameField.h"

GameField::GameField(int fieldSize):

fieldHeight(fieldSize),

fieldWidth(fieldSize),

objectsCount(0)

{

field = new Unit\*\* [fieldSize];

for (int i=0; i<fieldSize; i++){

field[i] = new Unit\* [fieldSize];

for (int j=0; j<fieldSize; j++){

field[i][j] = nullptr;

}

}

}

GameField::GameField(int fieldHeight, int fieldWidth):

fieldHeight(fieldHeight),

fieldWidth(fieldWidth),

objectsCount(0) {

field = new Unit \*\*[fieldHeight];

for (int i = 0; i < fieldHeight; i++) {

field[i] = new Unit \*[fieldWidth];

for (int j = 0; j < fieldWidth; j++) {

field[i][j] = nullptr;

}

}

}

void GameField::deleteObject(int x, int y) {

if (field[y][x]) {

objectsCount--;

delete field[y][x];

field[y][x] = nullptr;

}

}

void GameField::addObject(Unit \*object, int x, int y) {

bool isInBorder = x < fieldWidth && y < fieldHeight && x >= 0 && y >= 0;

if (isInBorder && !field[y][x] && objectsCount < maxObjectsCount && !object->isOnField){

field[y][x] = object;

objectsCount++;

object->position = Point(x, y);

} else{

std::cout << "Impossible to add Object " << object << " to field." << std::endl;

}

}

void GameField::deleteObject(Unit \*object) {

deleteObject(object->position.x, object->position.y);

}

void GameField::moveObject(const Point &p1, const Point &p2) {

if (field[p1.y][p1.x] && !field[p2.y][p2.x]){

field[p2.y][p2.x] = field[p1.y][p1.x];

field[p2.y][p2.x]->position = p2;

field[p1.y][p1.x] = nullptr;

} else{

std::cout << "Impossible to move object. Coord " << p1.x << ' ' << p2.y << " is empty";

}

}

void GameField::moveObject(Unit \*object, const Point &p2) {

Point p1 = object->getPosition();

moveObject(p1, p2);

}

GameField::GameField(const GameField &other) :

fieldHeight(other.fieldHeight),

fieldWidth(other.fieldWidth),

maxObjectsCount(other.maxObjectsCount)

{

field = new Unit\*\* [fieldHeight];

for (int i=0; i<fieldHeight; i++){

field[i] = new Unit\* [fieldWidth];

for (int j=0; j<fieldWidth; j++){

if (other.field[i][j])

field[i][j] = new Unit(other.field[i][j]);

else

field[i][j] = nullptr;

}

}

}

void GameField::deleteObject(const Point &point) {

deleteObject(point.x, point.y);

}

int GameField::getHeight() const{

return fieldHeight;

}

int GameField::getWidth() const{

return fieldWidth;

}

Unit \*GameField::getObject(const Point &p)const {

if (p.x < fieldWidth && p.y < fieldHeight)

return field[p.y][p.x];

return nullptr;

}

GameField::GameField(GameField &&other) :

field (other.field),

fieldHeight (other.fieldHeight),

fieldWidth (other.fieldWidth),

maxObjectsCount(other.maxObjectsCount),

objectsCount(other.objectsCount)

{

field = other.field;

other.field = nullptr;

}

GameField::~GameField(){

for(int i =0; i < fieldHeight; i++){

delete []field[i];

}

delete []field;

}