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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ООП»

Тема: "Создание классов, конструкторов классов, методов классов, наследование».

Студент гр. 8304 ____ Самакаев Д.И.

Преподаватель ____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы

Реализовать набор классов игрового поля и юнитов, классы должны удовлетворять следующим требованиям: создание поля произвольного размера, контроль максимального количества объектов на поле, возможность добавления и удаления объектов, их перемещение, копирование поля со всеми объектами, юниты должны иметь общий интерфейс.

Постановка задачи

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера
- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)
- Для хранения запрещается использовать контейнеры из stl

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа(например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Баллы за лаб. работу (* отмечает необязательные пункты)

Ход работы.

Были разработаны классы поля и юнитов, реализованы возможности помещения юнита на поле, его удаления, перемещения, копирования поля.

Класс поля Field хранит свои размеры, массив символов и указатель на массив указателей на объекты типа Unit. Также в классе Field реализованы методы: вывод поля на экран, добавление юнита на поле, удаление юнита с поля, перемещение юнита по полю, была перегружена операция присваивания для успешного копирования поля со всеми его элементами.

Класс Unit и отнаследованные от него классы хранят информацию о юнитах, имеют такие характеристики как атака, броня, здоровье и имя.

Вывод

Был разработан и реализован набор классов, позволяющий осуществлять действия над юнитами и полем.

ПРИЛОЖЕНИЕ А.

Исходный код.

Unit.h:

```
#pragma once
#include <iostream>
#include <memory>

class Unit {
public:
	Unit() {
		this->unit_name = ' ';

		this->health = 0;
		this->armor = 0;
		this->attack = 0;

		x = 0;
		y = 0;
	}

	Unit(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
name) {
		this->unit_name = name;

		this->health = health;
		this->armor = armor;
		this->attack = attack;

		x = coord_x;
		y = coord_y;
	}

	~Unit() {

	}

	void change_health(int new_health) {
		if (new_health <= 0)
			this->~Unit();
		else health = new_health;
	}

	void change_position(size_t new_x, size_t new_y) {
		x = new_x;
		y = new_y;
	}

	void on_damage_taken(size_t damage) {
		if (damage <= armor)
			change_armor(armor - damage);
		else {
			change_armor(0);
			change_health(health + armor - damage);
		}
	}

	void change_armor(int new_armor) {
		if (new_armor >= 0)
			attack = new_armor;
		else attack = 0;
	}

	void change_attack(int new_attack) {
		if (new_attack >= 0)
			attack = new_attack;
		else attack = 0;
```

```
	}

	char get_unit_name() {
		return unit_name;
	}

	size_t get_x() {
		return x;
	}

	size_t get_y() {
		return y;
	}
private:
	char unit_name;

	size_t health;
	size_t armor;
	size_t attack;

	size_t x;
	size_t y;

};

class Vehicle : public Unit {
public:
	Vehicle(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y,
char unit_name)
		: Unit(health, armor, attack, coord_x, coord_y, unit_name) {
	}
};

class Plane : public Vehicle {
public:
	Plane(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
unit_name)
		: Vehicle(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Tank : public Vehicle {
public:
	Tank(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
unit_name)
		: Vehicle(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Humans : public Unit {
public:
	Humans(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y,
char unit_name)
		: Unit(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Sniper : public Humans {
public:
	Sniper(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y,
char unit_name)
		: Humans(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Medic : public Humans {
```

```
public:
	Medic(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
unit_name)
		: Humans(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Beasts : public Unit {
public:
	Beasts(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y,
char unit_name)
		: Unit(health, armor, attack, coord_x, coord_y, unit_name) {
		beasts_name = unit_name;
	}

	char get_beasts_name() {
		return beasts_name;
	}
private:
	char beasts_name;
};

class Dog : public Beasts {
public:
	Dog(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
unit_name)
		: Beasts(health, armor, attack, coord_x, coord_y, unit_name) {

	}
};

class Cat : public Beasts {
public:
	Cat(size_t health, size_t armor, size_t attack, size_t coord_x, size_t coord_y, char
unit_name)
		: Beasts(health, armor, attack, coord_x, coord_y, unit_name) {

	}
	 };
```

Field.h

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

class Field {
public:

	Field(size_t x, size_t y) {
		field_cells = new char[x * y];
		x_size = x;
		y_size = y;
		unit_field = new Unit * [x * y];
		for (size_t i = 0; i < x * y; i++) {
			field_cells[i] = ' ';
		}
	}

	~Field() {
		delete field_cells;
		for (size_t i = 0; i < this->x_size * this->y_size; i++) {
			if (unit_field[i] != nullptr)
				unit_field[i]->~Unit();
		}
		delete unit_field;
	}
```

```
	void add_unit(Unit* test) {
		field_cells[test->get_x() + x_size * test->get_y()] = test->get_unit_name();
		unit_field[test->get_x() + x_size * test->get_y()] = test;
	}

	void delete_unit(size_t x, size_t y) {
		field_cells[x + x_size * y] = ' ';
		unit_field[x + x_size * y]->~Unit();
		return;
	}

	void relocate_unit(size_t old_x, size_t old_y, size_t x, size_t y) {
		unit_field[old_x + x_size * old_y]->change_position(x, y);
		add_unit(unit_field[old_x + x_size * old_y]);
		delete_unit(old_x, old_y);

		field_cells[x + x_size * y] = unit_field[x + x_size * y]->get_unit_name();
	}

	void print_field() {
		std::cout << " ";
		for (size_t i = 0; i < this->x_size; i++)
			std::cout << "_";
		for (size_t i = 0; i < this->x_size * this->y_size; i++) {
			if (i % x_size == 0) {
				if (i != 0)
					std::cout << "|" << std::endl;
				else
					std::cout << std::endl;
				std::cout << "|";
			}
			std::cout << field_cells[i];
		}
		std::cout << "|" << "\n" << " ";
		for (size_t i = 0; i < this->x_size; i++)
			std::cout << "-";
		std::cout << std::endl;
	}

	Field& operator=(const Field& field) {
		this->~Field();
		x_size = field.x_size;
		y_size = field.y_size;
		field_cells = new char[x_size * y_size];
		unit_field = new Unit * [x_size * y_size];
		for (size_t i = 0; i < field.x_size * field.y_size; i++) {
			if (field.field_cells[i] != ' ') {
				unit_field[i] = new Unit(*field.unit_field[i]);
			}
			field_cells[i] = field.field_cells[i];
		}
		return *this;
	}
private:
	size_t x_size = 0;
	size_t y_size = 0;

	char* field_cells;

	Unit** unit_field;
	 };
```

Main.cpp:

```
#include "field.h"
#include "unit.h"
```

```
int main() {

	Field field = { 6, 6 };
	field.print_field();
	Vehicle vehicle{ 0,0,0,2,2,'V' };
	Humans human{ 0,0,0,1,2,'H' };
	Beasts beast{ 0,0,0,3,3,'B' };
	field.add_unit(&vehicle);
	field.print_field();

	Field field_cpy = { 4, 4 };
	field_cpy.add_unit(&human);
	field_cpy.print_field();
	field_cpy = field;
	field_cpy.add_unit(&beast);
	field_cpy.print_field();
	field_cpy.relocate_unit(3, 3, 5, 5);
	field_cpy.print_field();



	return 0;
	 }
```