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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Логическое разделение классов

Студент гр. 8304 _____ Порывай П.А

Преподаватель _____ Размочаева Н.В

Санкт-Петербург

2020

Цель работы

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Выполнение работы

1) Был реализован класс Command, являющийся базовым в паттерне “Команда” с виртуальной функцией для наследников этого класса, были созданы наследники, которые отвечают определенным действиям

2) Была реализована функция attack, которая атакует объект и перемещает на его место атакующего, если это не ”лучники”, функция также обновляет данные в базе, может вызывается через общий интерфейс – паттерн ”команда”

3) Реализована функция просмотра состояния базы, через которую выводятся расположения юнитов от данной базы,которые были созданы, уничтожены.

Пример работы программы

На рис. 1 показывается расположение юнитов на карте, а также список юнитов от базы, расположенной по координатам 2 2

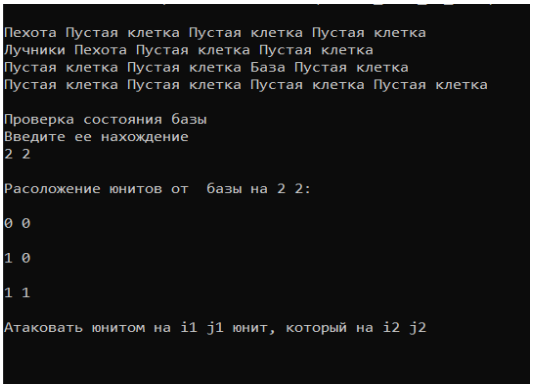

Рисунок 1

На рис.2 пример использования команды attack

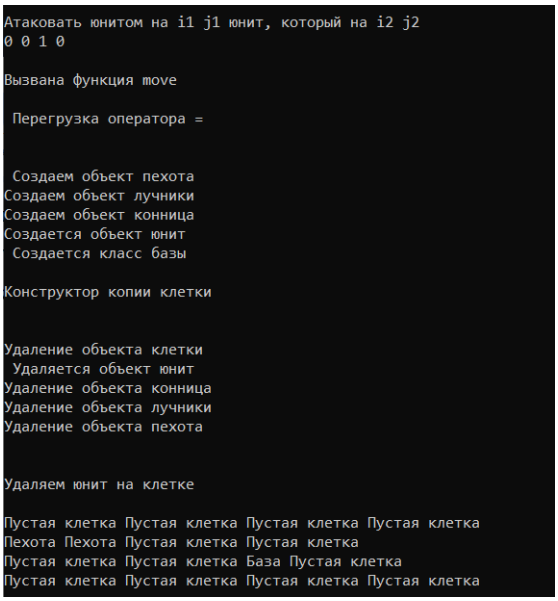

Рисунок 2

Выводы

Был реализован паттерн команда, методы для работы с базой, юнитами

Приложение А

Исходный код

Pattern_command.h

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

class Command {

public:
	virtual ~Command(){}
	virtual void execute() = 0;
protected:
	Command( playing_field* p ): p_f( p){}
	playing_field* p_f;
};

class inic_cell_Command : public Command
{

public:
	inic_cell_Command( playing_field *p):Command(p){}
	void execute() {


		std::cout << "\nВведите строку/столбец, куда разместить
юнит\n";


		int i, j;
		cin >> i >> j;
		p_f->inic_cell(i,j);
	}
};

class set_base_Command : public Command
{

public:
	set_base_Command(playing_field* p) :Command(p) {}
	void execute() {


		std::cout << "\nВведите строку/столбец, куда разместить
базу\n";


		int i, j;
		cin >> i >> j;
		p_f->set_base(i, j);
	}
};
```

```
class create_unit_of_base_Command : public Command
{

public:
	create_unit_of_base_Command(playing_field* p) :Command(p) {}
	void execute() {


		std::cout << "\nВведите строку/столбец, базы, которая
создает юнит,строку/столбец куда разместить юнит\n";


		int i_base, j_base,i_unit,j_unit;
		cin >> i_base >> j_base>>i_unit>>j_unit;
		p_f->create_unit_of_base(i_base , j_base , i_unit ,
j_unit);
	}


};

class output_field_Command :public Command {

public:
	output_field_Command(playing_field* p) :Command(p) {}
	void execute() {

		p_f->output_field();
	}
};


class move_Command :public Command {

public:
	move_Command(playing_field* p) :Command(p) {}
	void execute() {


		std::cout << "\nПередвинуть юнит куда i j/откуда i j \n";
//move(i1, j1, i2, j2);
		int i1, j1, i2, j2;
		cin >> i1 >> j1 >> i2 >> j2;
		p_f->move(i1, j1, i2, j2);
	}
};



class attack_Command :public Command {

public:
	attack_Command(playing_field* p) :Command(p) {}
	void execute() {//атака юнитом в i1 j1  другого юнита в i2 j2
```

```
            std::cout << "\nАтаковать юнитом на i1 j1 юнит, который на
i2 j2 \n";
            int i1, j1, i2, j2;
            cin >> i1 >> j1 >> i2 >> j2;
            p_f->attack(i1, j1, i2, j2);
      }
};



class state_base_Command :public Command {

public:
      state_base_Command(playing_field* p) :Command(p) {}
      void execute() {


            std::cout << "\nПроверка состояния базы \nВведите ее
нахождение\n";
            int i,j;
            cin >> i>>j;
            p_f->base_state(i, j);
      }

      };

      Playing_field.h



#pragma once
#include"cell_ancestors.h"
#include"landscape_and_inertobj.h"

cell init();//если поместить в playing_field.cpp, что-то изменится?
struct bases_location {

      int i = -1;
      int j = -1;

};


class playing_field {

      int max_obj;
      int height;
      int width;
      cell** p_f;

      bases_location bases[100];
      int i_bases = 0;

      int card_landscape[100][100];
      mountain mountains[100];
```

```
	swamp swamps[100];
	nettle nettles[100];
	flat flats[100];

	int i_mountains, i_swamps, i_nettles, i_flats;

public:


	playing_field(int height, int width, int** m);
	playing_field(const playing_field& ob2);//Конструктор копии


	void remove_obj(int i, int j);
	playing_field() {

		max_obj = 0;
		height = width = 0;
		p_f = nullptr;

		std::cout << "\nОбъект создан с помощью конструктора без параметров\n\n";

	}

	void inic_cell(int i, int j);//клетка

	//лаба 2
	void set_base(int i, int j) {

		p_f[i][j].write_in = 2;//устанавливаем базу

		bases[i_bases] = { i,j };
		card_landscape[i][j] = 2;
		i_bases++;

	}

	void create_unit_of_base(int i_base, int j_base, int i_unit, int j_unit);//создание юнита от базы
	//end lw2
	//в базе должны быть все расположения юнитов и при их перемещении они должны обновляться

	//Для след функций можно сделать проверку указателя p_f
	char* type_cell(int i, int j) {


		return p_f[i][j].get_type();

	}


	int health_cell(int i, int j) {

		return p_f[i][j].get_health();
```

```
	}

	int attack_cell(int i, int j) {

		return p_f[i][j].get_attack_force();

	}

	int drug_cell(int i, int j) {

		return p_f[i][j].get_paracetomol();
	}

	void output_field();
	int get_height() {
		return height;

	}

	int get_width() {

		return width;
	}

	//cell get_cell(int i,int j){

	//	cell ob;
	//	ob = p_f[i][j];

	//	return ob;//конструктор копии?
	//}

	char* get_type(int i, int j) {

		return p_f[i][j].get_type();
	}

	char* get_kind(int i, int j) {
		return p_f[i][j].get_kind();
	}
	void move(int i1, int j, int i2, int j2);

	playing_field operator=(playing_field& ob);

	~playing_field();

	void attack(int i1, int j1, int i2, int j2);//атака юнитом базы
или другого юнита
	void base_state(int i, int j);//Подкорректировать

};
```

Playing_field.cpp

```
#include"cell_ancestors.h"
```

```
#include"playing_field.h"


void playing_field::move(int i1, int j1, int i2, int j2) {// (далее
продумать менять ли,не доработано на конницу, лучников)

	std::cout << "\nВызвана функция move\n";

	if (card_landscape[i1][j1] != 3 ) {//? для 2

		p_f[i1][j1] = p_f[i2][j2];//вызывается конструктор копии;
		remove_obj(i2, j2);


		//не обновляется база
	}
	else
		std::cout << "\nВ заданной клетке болото, не возможно
переместиться сюда\n";

	//remove_obj(i2, j2);

}



void playing_field::remove_obj(int i, int j) {


	std::cout << "\nУдаляем юнит на клетке\n";
	p_f[i][j].remove();
	//не обновляется база
}


playing_field::~playing_field() {


	if (p_f) {

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

			delete[] p_f[i];//таким образом удаляется массив
объектов


		}

		delete p_f;
	}

	std::cout << "Объект поле удален\n";
```

```
}



void playing_field::output_field() {//поменять под 2 л



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


		std::cout << "\n";

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


			if (p_f[i][j].is_writein() == 1) {

				std::cout << p_f[i][j].get_type();
				std::cout << " ";
			}
			else if (p_f[i][j].is_writein() == 0) {

				std::cout << "Пустая клетка";
				std::cout << " ";


			}
			else if (p_f[i][j].write_in == 2 || p_f[i][j].write_in
== 3) {
				std::cout << "База";
				std::cout << " ";
			}


		}

	}

	std::cout << "\n";

}

void playing_field::inic_cell(int i, int j) {

	std::cout << "\nВызвана инициализация клетки\n";

	if (p_f[i][j].write_in == 0) {//Добавить проверку
		p_f[i][j] = init();
		p_f[i][j].write_in = 1;
	}
```

```
	else if (p_f[i][j].write_in == 1)
		std::cout << "\nВ клетке юнит\n";
	else if (p_f[i][j].write_in == 2 || p_f[i][j].write_in == 3)
		std::cout << "\nВ клетке база\n";

}



playing_field::playing_field(int height, int width, int** m) {

	this->height = height;
	this->width = width;


	max_obj = height * width;

	p_f = new cell * [height];

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

		p_f[i] = new cell[width];

	}
	i_mountains = i_swamps = i_nettles = i_flats = 0;

	for (int i = 0; i < height; i++)
		for (int j = 0; j < width; j++) {


			if (m[i][j] == 1) {//крапива
				card_landscape[i][j] = 1;
				nettles[i_nettles].i = i;
				nettles[i_nettles].j = j;
				i_nettles++;

			}
			else if (m[i][j] == 2) {//гора
				card_landscape[i][j] = 2;
				mountains[i_mountains].i = i;
				mountains[i_mountains].j = j;
				i_mountains++;
			}
			else if (m[i][j] == 3) {//болото
				card_landscape[i][j] = 3;
				swamps[i_swamps].i = i;
				swamps[i_swamps].j = j;
				i_swamps++;
			}
			else if (m[i][j] == 0) {//равнина
				card_landscape[i][j] = 0;
				flats[i_flats].i = i;
				flats[i_flats].j = j;
```

```
                        i_flats++;

                  }
            }
      std::cout << "Поле создано с помощью конструктора с параметрами
\n\n";
}


playing_field playing_field::operator=(playing_field& ob) {


      std::cout << "\nВызвана перегрузка оператора = для
playing_field\n";

      height = ob.height;
      width = ob.width;
      max_obj = ob.max_obj;

      if (p_f != nullptr) {

            for (int i = 0; i < height; i++)
                  delete[] p_f[i];

            delete p_f;

      }

      p_f = new cell * [height];

      for (int i = 0; i < height; i++)
            p_f[i] = new cell[width];






      for (int i = 0; i < height; i++)

            for (int j = 0; j < width; j++) {
                  if (ob.p_f[i][j].is_writein() == 1)
                        p_f[i][j].set(ob.get_type(i, j), ob.get_kind(i,
j));


            }
      //p_f[i][j] = ob.get_cell(i,j);//здесь вызывается перегрузка и
конструктор копии cell

      return *this;
}


playing_field::playing_field(const playing_field& ob) {// для =  он
условный
```

```
	std::cout << "\nВызван конструктор копии поля\n";

	p_f = new cell * [1];
	p_f[0] = new cell[1];
}

void playing_field::create_unit_of_base(int i_base, int j_base, int
i_unit, int j_unit) {//подстроить под landscape

	if (p_f[i_base][j_base].write_in == 2 ||
p_f[i_base][j_base].write_in == 3) {

		if (p_f[i_unit][j_unit].write_in == 1 ||
p_f[i_unit][j_unit].write_in == 2 || p_f[i_unit][j_unit].write_in ==
3) {

			std::cout << "\nОшибка в данных невозможно разместить
юнит здесь\n";
			return;

		}


		if (card_landscape[i_unit][j_unit] != 3) {

			char string[100] = "\0";

			if (card_landscape[i_unit][j_unit] == 1)//равнина
крапива гора болото
				strcpy(string, "Лучники");

			if (card_landscape[i_unit][j_unit] == 2)
				strcpy(string, "Конница");

			else {
				if (strlen(string) > 0) {
					if
(p_f[i_base][j_base].create_unit_from_base(i_unit, j_unit) == true) {

						std::cout << "\nВ этой местности
нельзя разместить объект " << string << "\n";

						p_f[i_unit][j_unit] = init();

						card_landscape[i_unit][j_unit] = 1;
						p_f[i_unit][j_unit].write_in = 1;
						std::cout << "\nОт базы
инициализирован объект юнит(create_unit_of_base)\n";
					}
					else
						std::cout << "\nБаза не может создать
больше юнитов\n";
				}
				else {
```

```
                              if
(p_f[i_base][j_base].create_unit_from_base(i_unit, j_unit) == true) {


                                    card_landscape[i_unit][j_unit] = 1;

                                    p_f[i_unit][j_unit].write_in = 1;
                                    p_f[i_unit][j_unit] = init();


                                    std::cout << "\nОт базы
инициализирован объект юнит(create_unit_of_base)\n";

                              }
                              else
                                    std::cout << "\nБаза не может создать
больше юнитов\n";
                        }
                  }

            }
            else if (card_landscape[i_unit][j_unit] == 3) {
                  std::cout << "В этой клетке болото и ничего нельзя
разместить\n";
            }
      }
      else
            std::cout << "\nОшибка в данных, базы в указанной клетке
нет\n";



}//надо как то связать юнит с базой


void playing_field::attack(int i1, int j1, int i2, int j2) {//атака
юнитом в i1 j1  другого юнита в i2 j2

      if (p_f[i1][j1].write_in != 1 ) {
            std::cout << "\nВ 1 клетке нет, атакующего юнита\n";
      }
      else {

            if (p_f[i2][j2].write_in != 1) {

                  std::cout << "\nКлетка пуста или в ней база, атака
невозможна\n";
            }
            else {

                  int a, b;
                  int health = p_f[i2][j2].get_health() -
p_f[i1][j1].get_attack_force();

                  if (health <= 0) {
```

```
                        p_f[i2][j2].remove();

                        if (strcmp(p_f[i1][j1].get_type(), "Лучники") !=
0) {
                              move(i2, j2, i1, j1);

                              //если юнит от базы убрать его из списка в
базе

                              for (int k = 0; k < 100; k++) {

                                    if (bases[k].i != -1) {

                                          units_location* p =
p_f[bases[k].i][bases[k].j].get_units_location();


                                          for (int i = 0; i <
p_f[bases[k].i][bases[k].j].get_number_of_units(); i++)
                                                if (p[i].i == i2 && p[i].j
== j2) {
                                                      p[i].i = -1;
                                                      p[i].j = -1;//не
лучников обновляем в списке базы
                                                      a = i2;
                                                      b = j2;
                                                }


                                          for (int i = 0; i <
p_f[bases[k].i][bases[k].j].get_number_of_units(); i++)
                                                if (p[i].i == i1 && p[i].j
== j1) {
                                                      p[i].i = a;
                                                      p[i].j = b;//не
лучников обновляем в списке базы
                                                }




                                    }
                              }

                        }
                        //надо определить
                  }//можно добавить else и отнять здоровье

            }

      }

};
```

```
void playing_field::base_state(int i, int j) {


	if (p_f[i][j].write_in == 2 || p_f[i][j].write_in == 3) {

		p_f[i][j].display_units(i, j);


	}
	else {

		std::cout << "\nНа данной клетке нет базы\n";

	}

}
```

Main.cpp

```
#include"playing_field.h"
#include"example_Composite.h"
#include"pattern_command.h"
#include<vector>
int main()
{
	//lw1
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int a, b;
	std::cin >> a >> b;

	int** landscape = new int* [a];
	for (int i = 0; i < a; i++)
		landscape[i] = new int[b];
	for (int i = 0; i < a; i++)
		for (int j = 0; j < b; j++)
			landscape[i][j] = 0;

	playing_field p_f(a, b,landscape);

	vector<Command*> Commands;

	Commands.push_back(new set_base_Command(&p_f));
	Commands.push_back(new create_unit_of_base_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
	Commands.push_back(new create_unit_of_base_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
	Commands.push_back(new create_unit_of_base_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
	Commands.push_back(new state_base_Command(&p_f));
	Commands.push_back(new attack_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
	Commands.push_back(new attack_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
	Commands.push_back(new attack_Command(&p_f));
	Commands.push_back(new output_field_Command(&p_f));
```

```
Commands.push_back(new state_base_Command(&p_f));
Commands.push_back(new move_Command(&p_f));
Commands.push_back(new output_field_Command(&p_f));

for (int i = 0; i < Commands.size(); i++)
      Commands[i]->execute();

for (int i = 0; i < Commands.size(); i++)
      delete Commands[i];


return 0;

}
```