

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Объектно-Оrientированное программирование»
Тема: Создание классов

Студент гр. 3341

Пчелкин Н.И.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Целью работы является изучение основ объектно-ориентированного программирования, создание базовых классов для написания игры «Морской бой».

Для выполнения поставленной цели требуется:

- Изучить основные понятия ООП, особенности создания классов;
- Разработать архитектуру базовых классов корабля, поля и менеджера кораблей;
- Реализовать эти классы и связь между ними.

Задание.

Создать класс игры, который реализует следующий игровой цикл:

Начало игры

Раунд, в котором чередуются ходы пользователя и компьютерного врага.

В свой ход пользователь может применить способность и выполняет атаку.

Компьютерный враг только наносит атаку.

В случае проигрыша пользователь начинает новую игру

В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

Класс игры может знать о игровых сущностях, но не наоборот

Игровые сущности не должны сами порождать объекты состояния

Для управления самой игрой можно использовать обертки над командами

При работе с файлом используйте идиому RAII.

Выполнение работы.

В ходе выполнения работы были разработаны перечисленные ниже классы.

Class GameController – класс, контролирующий процесс игры. Отвечает за ввод команд, а также запускает игру. Имеет 2 метода:

- *startGame()* – метод, запускающий игру;
- *acceptCommand(ICommand* command)* – метод, принимающий

команды от пользователя через CLI.

Class CLIInput – класс порождающий объекты команд. Считывает необходимые данные от пользователя и передает в класс *GameController*. Имеет 1 метод:

readCommand() – метод, который выбирает по ключу необходимую команду и возвращает ее. Если такой команды не существует возвращает *nullptr*.

Class ICommand – абстрактный класс-интерфейс для каждой из возможных команд пользователя. Имеет 1 виртуальный метод:

- *virtual void execute(Game& game)* – метод, который будет отвечать за выполнение команды.

К нему реализовано 9 методов-команд. Их структура однообразна, ничего внутри себя кроме метода *execute* они не содержат. Со всеми вы можете ознакомиться в приложении «А».

Class Game – класс игры, со всей внутренней реализацией. Содержит в себе информацию о пользователе, компьютере и самой игре.

- *printMessage()* – метод, который выводит подаваемую в него строку;
- *placeShips()* – метод, отвечающий за расстановки кораблей игроком;
- *spawnOpp()* – метод, обновляющий поле врага, случайно

расставляющий корабли по его полю;

- *oppsTurn()* – метод, реализующий ход компьютерного оппонента;
- *gameOver()* – метод обрабатывающий ситуацию конца игры, после

поражения игроком предложит начать игру заново или же завершить программу;

- *printInfo()* – метод, выводящий информацию о составе флота обоих игроков;
 - *printSettingInfo ()* – метод, выводящий на экран информацию во время расстановки кораблей. Подсказывает, сколько кораблей осталось установить игроку;
 - *startGame()* – метод, запускающий игру;
 - *checkAbilities()* – метод, который выводит на экран информацию о том, какая сейчас доступна способность игроку (выводит номер способности);
 - *printInfoAbilities()* – метод, дающий справку о существующих способностях и их номерах;
 - *useAbility()* – метод, позволяющий использовать способность;
 - *attack ()* – метод, реализующий атаку игроком на вражеское поле, в случае попадания даст право следующего хода тому же игроку;
 - *printMyField ()* – метод, выводящий на экран поле игрока;
 - *printOppsField ()* – метод, выводящий на экран поле соперника;
 - *isRunning()* – метод, определяющий идет ли игра или же она закончилась;
 - *saveGame()* – метод, сохраняющий игру через состояние игры;
 - *loadGame()* – метод, загружающий игру через состояние игры;
- Class GameState* – класс, хранящий информацию об игроках, а также способный записывать эту информацию в текстовый документ. Имеет 6 методов:
- *saveGame ()* – записывает информацию об игре в текстовый документ “save.txt”;
 - *loadGame ()* – загружает информацию с файла “save.txt” и расшифровывает, чтобы передать игре;
 - *parseInfo ()* – метод, расшифровывающий информацию с файла;
 - *buildBattlefield ()* и *buildShipManager ()* – методы, которые расшифровывают карту поля и менеджера кораблей соответственно.

- *GetGameMap()* – метод, создающий карту игры из имеющегося состояния.

Struct Player – структура, в которой хранится поле игрока и его менеджер кораблей.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1

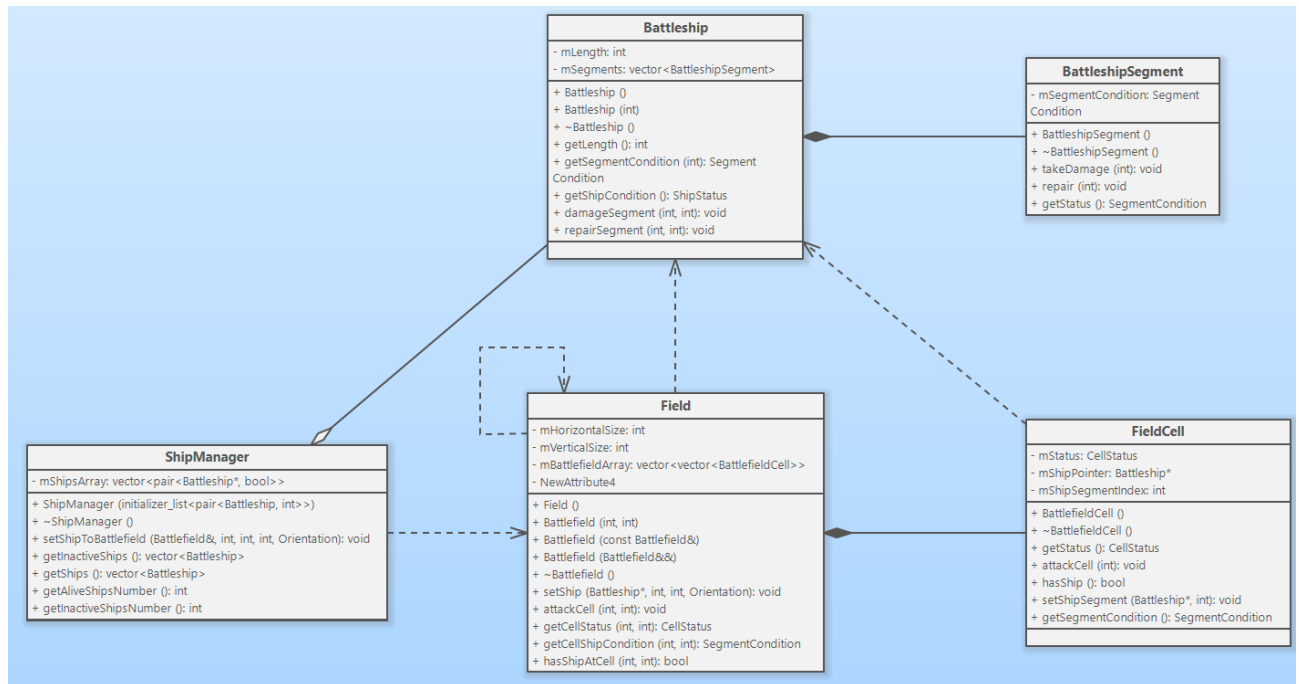


Рисунок 1 – диаграмма классов

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В результате выполнения лабораторной работы были изучены основы Объектно-ориентированного программирования на языке C++, базовые принципы построения архитектуры программ и создания классов. Были реализованы классы-интерфейсы способностей и их аргументов. Были реализованы классы способностей. Создан класс менеджера способностей. Была написана программа, проверяющая работоспособность разработанных классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Battleship.h

```
#include <iostream>
#include <vector>

#define HEALTHY_SEGMENT_REGISTER 'O'
#define DAMAGED_SEGMENT_REGISTER 'X'
#define DESTROYED_SEGMENT_REGISTER '.'
#define EMPTY_REGISTER '*'
#define MISS_REGISTER ' '
#define UNSIGNED_REGISTER '/'

enum Orientation {
    VERTICAL,
    HORIZOTNTAL
};

enum SegmentStatus{
    HEALTHY,
    DAMAGED,
    DESTROYED
};

enum CellStatus {
    UNNKOWN,
    EMPTY,
    SHIP,
    UNSIGNED
};

class Ship
{
public:
    class Segment {
```

```

private:
    SegmentStatus status = SegmentStatus::HEALTHY;
public:
    SegmentStatus check() const {
        return status;
    }
    void take_damage() {
        if (status == SegmentStatus::HEALTHY)
            status = SegmentStatus::DAMAGED;
        else if (status == SegmentStatus::DAMAGED)
            status = SegmentStatus::DESTROYED;
    }

};

Ship(int length) :length(length) {
    segments = new Segment[length];
    for (int i = 0; i < length; i++) {
        segments[i] = Segment();
    }
}

int size() {
    return length;
}

bool destroyed(){
    for (int i = 0; i < length; i++)
        if (segments[i].check() != SegmentStatus::DESTROYED)
            return false;
    return true;
}

Segment* get_segment(int index) {
    return &segments[index];
}

private:
    int length;
    Segment* segments = NULL;
};

```



```

class Battlefield {
    class Cell {
    private:
        CellStatus state = CellStatus::EMPTY;
        Ship::Segment* ptr_ship = nullptr;
        bool was_attacked = false;
    public:
        CellStatus check() {
            return state;
        }
        void hit() {
            if (was_attacked) {
                std::cout << "You have already attacked this
cell!\n";

                return;
            }
            switch (state)
            {
            case CellStatus::EMPTY:
                was_attacked = true;
                break;
            case CellStatus::SHIP:
                if (ptr_ship->check() == SegmentStatus::DESTROYED)
                    std::cout << "You have already destroyed
segment in this cell\n";
                else
                    ptr_ship->take_damage();
            }
        }
        void set_segment(Ship::Segment* segment) {
            ptr_ship = segment;
            state = CellStatus::SHIP;
        }
        Ship::Segment* get_segment() {
            return ptr_ship;
        }
        bool checked() {
            return was_attacked;
        }
    }
};

```

```

        };
private:
    Cell** field;
    int length;
    int width;
public:
    Battlefield(const int length, const int width) {
        this->length = length;
        this->width = width;
        field = new Cell*[length];
        for (int i = 0; i < length; i++) {
            field[i] = new Cell[width];
            for (int j = 0; j < width; j++)
                field[i][j] = Cell();
        }
    }

    Battlefield(const Battlefield& copy) :Battlefield(copy.length,
copy.width) {}

    Battlefield(Battlefield&& moved) {
        length = std::move(moved.length);
        width = std::move(moved.width);
        field = std::move(moved.field);
    }

    Battlefield& operator = (const Battlefield& copy) {
        if (&copy != this) {
            length = copy.length;
            width = copy.width;
            field = new Cell * [length];
            for (int i = 0; i < length; i++) {
                field[i] = new Cell[width];
                for (int j = 0; j < width; j++)
                    field[i][j] = Cell();
            }
        }
        return *this;
    }

    Battlefield& operator = (Battlefield&& moved) {
        if (&moved != this) {
            length = std::move(moved.length);

```

```

        width = std::move(moved.width);
        field = std::move(moved.field);
    }
    return *this;
}

~Battlefield() {
    for (int i = 0; i < length; i++)
        delete[] field[i];
    delete[] field;
}

CellStatus check(int x, int y) const {
    if (x >= 0 and y >= 0 and x < length and y < width)
        return field[x][y].check();
    else {
        return CellStatus::UNSIGNED;
    }
}

bool scanner(int x, int y) const {
    for (int i = -1; i < 2; i++)
        for (int j = -1; j < 2; j++)
            if (check(x + i, y + j) == CellStatus::SHIP)
                return true;
    return false;
}

bool set_ship(Ship& ship, int x, int y, Orientation orient) {
    int horizontal_offset, vertical_offset;
    if (orient == Orientation::VERTICAL) {
        horizontal_offset = 0;
        vertical_offset = ship.size() - 1;
    }
    else {
        horizontal_offset = ship.size() - 1;
        vertical_offset = 0;
    }
    if (scanner(x, y) or scanner(x+vertical_offset,
y+horizontal_offset) or check(x, y) == UNSIGNED_REGISTER or
check(x+vertical_offset, y+horizontal_offset) == UNSIGNED_REGISTER) {
        std::cout << "Can't place the ship here!\n";
    }
}

```

```

        return false;
    }
    for (int i = x; i <= x + vertical_offset; i++)
        for (int j = y; j <= y + horizontal_offset; j++)
            field[i][j].set_segment(ship.get_segment(i+j-
(x+y)));

    return true;
}

void attack(int x, int y) {
    if (check(x, y) != UNSIGNED_REGISTER)
        field[x][y].hit();
    else
        std::cout << "Can't attack this cell!\n";
}

void print() const {
    std::cout << "  ";
    for (int i = 0; i < width; i++)
        std::cout << i << ' ';
    std::cout << '\n';
    for (int i = 0; i < length; i++) {
        std::cout << i << ' ';
        for (int j = 0; j < width; j++) {
            char cell_info;
            if (field[i][j].check() == CellStatus::SHIP) {
                switch (field[i][j].get_segment()->check()) {
                    case SegmentStatus::HEALTHY:
                        cell_info = HEALTHY_SEGMENT_REGISTER;
                        break;
                    case SegmentStatus::DAMAGED:
                        cell_info = DAMAGED_SEGMENT_REGISTER;
                        break;
                    case SegmentStatus::DESTROYED:
                        cell_info = DESTROYED_SEGMENT_REGISTER;
                        break;
                }
            }
            else if (field[i][j].checked())
                cell_info = MISS_REGISTER;
            else

```

```

        cell_info = EMPTY_REGISTER;
        std::cout << cell_info << ' ';
    }
    std::cout << '\n';
}
}
};

class ShipManager {
private:
    std::vector<Ship> passive_ships[4];
    std::vector<Ship> active_ships[4];
public:
    ShipManager(int number_of_ships[4]) {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < number_of_ships[i]; j++)
                passive_ships[i].push_back(Ship(i + 1));
    }

    void set_ship(Battlefield& field, int length, int x, int y,
Orientation orient) {
        if (length < 1 and length > 4) {
            std::cout << "Wrong length for the ship!\n";
            return;
        }
        length--;
        if (passive_ships[length].size() == 0) {
            std::cout << "There is no more ships of this
length!\n";
            return;
        }
        if (field.set_ship(passive_ships[length].back(), x, y,
orient)) {
            active_ships[length].push_back(passive_ships[length].back());
            passive_ships[length].pop_back();
        }
    }
}

```

```

bool end_of_setting() const{
    for (int i = 0; i < 4; i++)
        if (passive_ships[i].size() != 0)
            return false;
    return true;
}

void setting_info() {
    for (int i = 0; i < 4; i++)
        std::cout << "i: " << passive_ships[i].size() <<
std::endl;
}

void battle_info(bool print=true) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < active_ships[i].size(); j++)
            if (active_ships[i].at(j).destroyed())

active_ships[i].erase(active_ships[i].begin()+j);
            if(print)
                std::cout << "i: " << active_ships[i].size() <<
std::endl;
        }
    }

bool game_over() {
    battle_info(false);
    for (int i = 0; i < 4; i++)
        if (active_ships[i].size() != 0)
            return false;
    return true;
}

```

};ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Тестирование написанных классов было выполнено в виде небольшой программы, которая создаёт объекты класса Battleship, Battlefield и ShipManager, добавляет в менеджер кораблей корабли, размещает их на поле, а затем – с помощью отладочных функций – выводит их на экран.

```
int main() {
    int m, n;
    std::cin >> m >> n;
    Battlefield field1 = Battlefield(m, n);
    field1.print();
    int a[4];
    for (int i = 0; i < 4; i++)
        std::cin >> a[i];
    ShipManager manager = ShipManager(a);
    while (!manager.end_of_setting()) {
        std::cout << "=====\n";
        int length;
        char o;
        std::cin >> length >> m >> n >> o;
        Orientation orient;
        if (o == 'h')
            orient = Orientation::HORIZOTNTAL;
        else if (o == 'v')
            orient = Orientation::VERTICAL;
        manager.set_ship(field1, length, m, n, orient);
        manager.setting_info();
        field1.print();
    }
    while (!manager.game_over()) {
        std::cout << "=====\n";
        std::cin >> m >> n;
        field1.attack(m, n);
        field1.print();
        manager.battle_info();
    }
}
```

```

    return 0;
}

```

Вывод:

```

10 10
  0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * * * * * *
4 * * * * * * * * *
5 * * * * * * * * *
6 * * * * * * * * *
7 * * * * * * * * *
8 * * * * * * * * *
9 * * * * * * * * *
1 1 0 0
=====
1 0 0 h
i: 0
i: 1
i: 0
i: 0
  0 1 2 3 4 5 6 7 8 9
0 O * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * * * * * *
4 * * * * * * * * *
5 * * * * * * * * *
6 * * * * * * * * *
7 * * * * * * * * *
8 * * * * * * * * *
9 * * * * * * * * *
=====
2 2 2 h
i: 0
i: 0

```



```

i: 0
i: 0
  0 1 2 3 4 5 6 7 8 9
0 O * * * * * * * *
1 * * * * * * * *
2 * * O O * * * *
3 * * * * * * * *
4 * * * * * * * *
5 * * * * * * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
9 * * * * * * * *

```

=====

```

0 0
  0 1 2 3 4 5 6 7 8 9
0 X * * * * * * * *
1 * * * * * * * *
2 * * O O * * * *
3 * * * * * * * *
4 * * * * * * * *
5 * * * * * * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
9 * * * * * * * *

```

i: 1

i: 1

i: 0

i: 0

=====

```

0 0
  0 1 2 3 4 5 6 7 8 9
0 . * * * * * * * *
1 * * * * * * * *
2 * * O O * * * *
3 * * * * * * * *
4 * * * * * * * *
5 * * * * * * * *

```

```

6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
i: 0
i: 1
i: 0
i: 0

```

=====

```

2 2
  0 1 2 3 4 5 6 7 8 9
0 . * * * * *
1 * * * * *
2 * * X O * * * * *
3 * * * * *
4 * * * * *
5 * * * * *
6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
i: 0
i: 1
i: 0
i: 0

```

=====

```

3 3
  0 1 2 3 4 5 6 7 8 9
0 . * * * * *
1 * * * * *
2 * * X O * * * * *
3 * * * * *
4 * * * * *
5 * * * * *
6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
i: 0

```

```

i: 1
i: 0
i: 0
=====

2 3
  0 1 2 3 4 5 6 7 8 9
0 . * * * * * * * * *
1 * * * * * * * * *
2 * * X X * * * * *
3 * * * * * * * * *
4 * * * * * * * * *
5 * * * * * * * * *
6 * * * * * * * * *
7 * * * * * * * * *
8 * * * * * * * * *
9 * * * * * * * * *

i: 0
i: 1
i: 0
i: 0
=====

2 3
  0 1 2 3 4 5 6 7 8 9
0 . * * * * * * * * *
1 * * * * * * * * *
2 * * X . * * * * *
3 * * * * * * * * *
4 * * * * * * * * *
5 * * * * * * * * *
6 * * * * * * * * *
7 * * * * * * * * *
8 * * * * * * * * *
9 * * * * * * * * *

i: 0
i: 1
i: 0
i: 0
=====

2 2

```

	0	1	2	3	4	5	6	7	8	9
0	.	*	*	*	*	*	*	*	*	*
1	*	*	*	*	*	*	*	*	*	*
2	*	*	.	.	*	*	*	*	*	*
3	*	*	*		*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*

i: 0

i: 0

i: 0

i: 0