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

Студент гр. 3381 Щеглов М.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Цель работы - создать системы способностей для игры "Морской бой", включая реализацию трёх способностей: двойной урон, сканер и обстрел, которые игрок может применять. Менеджер хранит способности в случайном порядке и добавляет новые при уничтожении кораблей. Реализована обработка исключений для ситуаций, таких как отсутствие доступных способностей, некорректное размещение корабля и атака за пределы поля.

Задание.

A) Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

1) Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

2) Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

3) Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

B) Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

C) Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

D) Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

1) Попытка применить способность, когда их нет

2) Размещение корабля вплотную или на пересечении с другим кораблем

3) Атака за границы поля

Примечания:

- Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
- Не должно быть явных проверок на тип данных

Выполнение работы

В файле **abilities.h** определяются базовые интерфейсы для способностей, которые могут быть применены в игре.

1. **Ability**:
    - `void useAbility(GameField& field, ShipManager& manager)` — абстрактный метод, который будет реализован в каждом конкретном классе-наследнике и применяется для выполнения действия способности.
    - `std::string getName() const` — метод для получения имени способности, что позволяет идентифицировать каждую из них.
2. **SetCoordinates**:
    - Наследует класс Ability и добавляет метод `setCoordinates(int x, int y)`.
    - Предназначен для способностей, которые требуют установку координат перед выполнением.

Класс **AbilityManager** управляет доступными способностями игрока, обеспечивая хранение, использование и динамическое добавление новых способностей по мере уничтожения вражеских кораблей.

1. Частные поля:
    - `std::deque<std::unique_ptr<Ability>> abilities`: очередь способностей, представленная уникальными указателями на объекты типа Ability.

- `int abilityCount`: целочисленное поле, содержащее текущее количество доступных способностей в очереди.

2. Конструктор `AbilityManager()`:
   - Инициализирует очередь с базовым набором способностей в случайном порядке.
3. Публичные методы:
   - `Const std::deque<std::unique_ptr<Ability>>& getAbilities() const` - Возвращает ссылку на текущую очередь способностей.
   - `int getAbilityCount() const` - Возвращает количество доступных способностей.
   - `void useAbility(GameField& field, ShipManager& manager, int x = 0, int y = 0)`- применяет первую способность из очереди, если она доступна. Перед применением проверяется, поддерживает ли способность интерфейс SetCoordinates; если поддерживает, устанавливаются заданные координаты. После использования способность удаляется из очереди и счётчик уменьшается.
   - `void addRandomAbility(int shipDestroyed)` - добавляет случайную способность в очередь, если был уничтожен корабль. Принимает параметр shipDestroyed, который сигнализирует об уничтожении корабля (значение 1) или его отсутствии (значение 0).

Класс **DoubleDamage**: способность наносит двойной урон по заданным координатам. При успешном применении атака дважды поражает один и тот же сегмент корабля, что может привести к его мгновенному уничтожению.

Методы:

- `setCoordinates(int x, int y)` — устанавливает координаты клетки, на которую будет направлен двойной урон.

- `useAbility(GameField& field, ShipManager& manager)` — вызывает метод attack для field дважды по одним и тем же координатам, что соответствует нанесению двойного урона.

Класс **Scanner**: способность, которая позволяет сканировать небольшую область 2x2 на поле вокруг заданной точки и проверять наличие сегментов корабля в этом диапазоне. Сканирование не меняет статус клеток, но предоставляет игроку дополнительную информацию.

Методы:

- `setCoordinates(int x, int y)` — задает координаты центральной точки для сканирования.
- `useAbility(GameField& field, ShipManager& manager)` — проверяет клетки в радиусе 1 клетки вокруг (x, y), выводя сообщения, если в какой-либо из клеток находится сегмент корабля.

Класс **GunBlaze**: описание: Способность, которая наносит урон по случайному сегменту случайного корабля. Координаты атаки генерируются случайным образом, поэтому игрок не контролирует, какой именно сегмент корабля будет поврежден. Эта способность также не меняет видимый статус клеток на поле.

Методы:

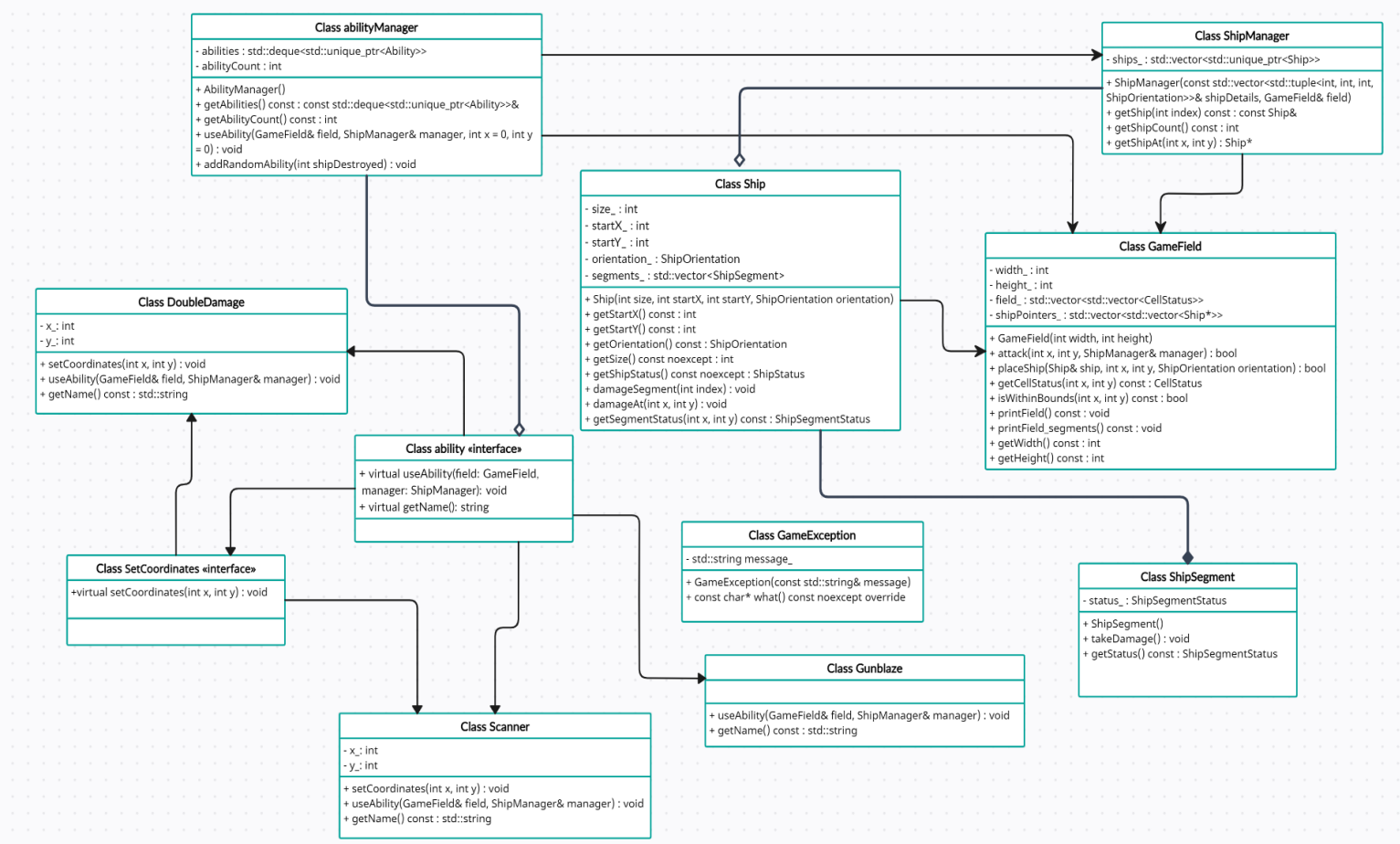
- `useAbility(GameField& field, ShipManager& manager)` — случайным образом выбирает координаты для атаки, используя генератор случайных чисел. Метод attack проверяет наличие корабля на случайно выбранной клетке и наносит урон, если сегмент корабля найден.

Файл *exceptions.h* содержит набор классов-исключений, которые используются для обработки различных ошибок. Все классы-исключения наследуются от базового класса GameException, который, в свою очередь, является наследником стандартного класса std::exception.

Классы в exceptions.h:

1. ***GameException**:*
    - Базовый класс для всех исключений в игре.
    - Содержит поле для хранения сообщения об ошибке и переопределённый метод ***what()***, который возвращает это сообщение.
2. ***OutOfBoundsAttackException**:*
    - Исключение, которое выбрасывается при попытке атаковать за пределами игрового поля.
3. ***InvalidShipSizeException**:*
    - Исключение для случаев, когда передаётся некорректный размер корабля (меньше минимального или больше максимального).
4. ***InvalidSegmentIndexException**:*
    - Исключение, которое выбрасывается при попытке доступа к несуществующему сегменту корабля по индексу.
    - Сообщение по умолчанию: "Segment index is out of range!"
5. ***CoordinatesOutOfBoundsException**:*
    - Исключение, которое выбрасывается при попытке получить статус клетки, координаты которой выходят за пределы поля.
6. ***ShipPlacementException**:*
    - Исключение для случаев, когда корабль не может быть размещён на игровом поле.
    - Принимает сообщение, указывающее, почему размещение не удалось.

Ниже представлена UML-диаграмма реализованных в данной работе классов:



Разработанный программный код см. в приложении А.

Тестирование программы см. в приложении Б.

Выводы

В ходе выполнения второй лабораторной работы была успешно реализована система способностей, включая три уникальные способности: двойной урон, сканер и обстрел. Также был создан класс менеджера способностей, который управляет очередью случайных способностей и их применением. Кроме того, была реализована система исключений для обработки некорректных ситуаций.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: exceptions.h

```
#ifndef EXCEPTIONS_H
#define EXCEPTIONS_H

#include <exception>
#include <string>

class GameException : public std::exception {
public:
    explicit GameException(const std::string& message) : message_(message) {}

    const char* what() const noexcept override {
        return message_.c_str();
    }

private:
    std::string message_;
};

// СоБаКа
class sobaka : public GameException {
public:
    sobaka() : GameException("СОБАКА (dog)") {}
};

// Исключение: попытка разместить корабль вплотную к другому кораблю
class ShipOverlapException : public GameException {
public:
```

```
        ShipOverlapException() : GameException("Ошибка: попытка разместить корабль вплотную или на пересечении с другим кораблем!") {}
    };

    // Исключение: попытка атаки за границы поля
    class OutOfBoundsAttackException : public GameException {
    public:
        OutOfBoundsAttackException() : GameException("Ошибка: попытка атаки за пределы поля!") {}
    };

    // Исключение для некорректного размера корабля
    class InvalidShipSizeException : public GameException {
    public:
        InvalidShipSizeException() : GameException("Размер корабля должен быть от 1 до 4") {}
    };

    // Исключение для недопустимого индекса сегмента корабля
    class InvalidSegmentIndexException : public GameException {
    public:
        InvalidSegmentIndexException() : GameException("Индекс сегмента корабля за границей") {}
    };

    // Исключение для недопустимого индекса корабля
    class InvalidShipIndexException : public GameException {
    public:
        InvalidShipIndexException() : GameException("Индекс корабля за границей") {}
    };

    // Исключение для некорректных размеров игрового поля
    class InvalidFieldSizeException : public GameException {
```

```
    public:
        InvalidFieldSizeException()    :    GameException("Неверные
размеры поля, прямоугольные или отрицательные") {}
    };

    // Исключение для координат, выходящих за границы игрового
поля
    class CoordinatesOutOfBoundsException : public GameException
{
    public:
        CoordinatesOutOfBoundsException()                          :
GameException("Coordinates are out of bounds!") {}
    };

    #endif
```

Название файла: abilityManager.h

```
    #ifndef ABILITY_MANAGER_H
    #define ABILITY_MANAGER_H

    #include <deque>
    #include <algorithm>
    #include <memory>
    #include <random>

    #include "abilities.h"

    class AbilityManager {
    private:
        std::deque<std::unique_ptr<Ability>>   abilities;       //
Очередь способностей
        int abilityCount;

    public:
        AbilityManager();
```

```
        // Получение очереди способностей
        const std::deque<std::unique_ptr<Ability>>& getAbilities()
const;

        // Получение количества способностей
        int getAbilityCount() const;

        // Применение первой способности из очереди
        void useAbility(GameField& field, ShipManager& manager,
int x = 0, int y = 0);

        // Добавление случайной способности в очередь
        void addRandomAbility(int shipDestroyed);
    };

    #endif // ABILITY_MANAGER_H
```

Название файла: abilityManager.cpp

```
#include "abilityManager.h"
#include "DoubleDamage.h"
#include "Scanner.h"
#include "GunBlaze.h"
#include <iostream>
#include <random>
#include <algorithm>

// Конструктор: создаем случайную очередь способностей
AbilityManager::AbilityManager() {
    std::random_device rd;
    std::mt19937 gen(rd());

    abilities.push_back(std::make_unique<DoubleDamage>());
    abilities.push_back(std::make_unique<Scanner>());
    abilities.push_back(std::make_unique<GunBlaze>());
```

```
        std::shuffle(abilities.begin(), abilities.end(), gen);

        abilityCount = abilities.size();
    }

    // Получение очереди способностей (константная ссылка)
    const                    std::deque<std::unique_ptr<Ability>>&
AbilityManager::getAbilities() const {
        return abilities;
    }

    // Получение количества способностей
    int AbilityManager::getAbilityCount() const {
        return abilityCount;
    }

    // Применение первой способности из очереди
    void       AbilityManager::useAbility(GameField&       field,
ShipManager& manager, int x, int y) {
        if (!abilities.empty()) {
            // Проверка, требуется ли установка координат для
способности
            SetCoordinates*            coordAbility            =
dynamic_cast<SetCoordinates*>(abilities.front().get());
            if (coordAbility) {
                coordAbility->setCoordinates(x, y);  // Установка
координат, если это поддерживается
            }
            abilities.front()->useAbility(field, manager);
            abilities.pop_front();
            abilityCount = abilities.size();
        } else {
            std::cout   <<   "Нет   доступных   способностей   для
использования!" << std::endl;
```

```
        }
    }

    // Добавление случайной способности в очередь, если корабль
уничтожен
    void AbilityManager::addRandomAbility(int shipDestroyed) {
        if (shipDestroyed == 1) {
            std::random_device rd;
            std::mt19937 gen(rd());
            std::uniform_int_distribution<> dis(0, 2);

            int randomNum = dis(gen);
            switch (randomNum) {
                case 0:

abilities.push_back(std::make_unique<DoubleDamage>());
                    break;
                case 1:

abilities.push_back(std::make_unique<Scanner>());
                    break;
                case 2:

abilities.push_back(std::make_unique<GunBlaze>());
                    break;
            }
            std::cout << "Добавлена новая случайная способность!"
<< std::endl;
            abilityCount = abilities.size();
        }
    }
```

Файл abilities.h

```
    #ifndef ABILITIES_H
    #define ABILITIES_H
```

```
#include <string>
#include "field.h"
#include "shipManager.h"

class Ability {
public:
    virtual ~Ability() = default;
    virtual void useAbility(GameField& field, ShipManager&
manager) = 0;
    virtual std::string getName() const = 0;
};

class SetCoordinates : public Ability {
public:
    virtual void setCoordinates(int x, int y) = 0;
};

#endif
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Название файла: main.cpp

```
#include "ship.h"
#include "shipManager.h"
#include "field.h"
#include "DoubleDamage.h"
#include "Scanner.h"
#include "GunBlaze.h"
#include "abilityManager.h"
#include <iostream>
#include <tuple>
#include <vector>
#include <memory>

int main() {
    // Инициализация параметров кораблей: {size, startX, startY,
orientation}
    std::vector<std::tuple<int, int, int, ShipOrientation>>
shipDetails = {
        {4, 1, 8, ShipOrientation::Horizontal},
        {1, 1, 1, ShipOrientation::Horizontal},
```

```
    {3, 7, 3, ShipOrientation::Vertical}
};

GameField field(10, 10);

ShipManager manager(shipDetails, field);

field.printField();
std::cout << std::endl;

// Проводим несколько атак по полю
field.attack(0, 0, manager);
field.attack(1, 0, manager);
field.attack(2, 0, manager);

field.attack(1, 1, manager);
field.attack(1, 1, manager);

field.attack(1, 8, manager);

// Использование способностей напрямую
DoubleDamage doubleDamage;
doubleDamage.setCoordinates(7, 3);
doubleDamage.useAbility(field, manager);

field.printField();
std::cout << std::endl;

Scanner scanner;
scanner.setCoordinates(0, 8);
scanner.useAbility(field, manager);

GunBlaze gunBlaze;
gunBlaze.useAbility(field, manager);



field.printField();
std::cout << std::endl;
```

```
        // Создаем менеджер способностей
        AbilityManager abilityManager;

        // Используем первую способность из очереди менеджера способностей
        if (!abilityManager.getAbilities().empty()) {
            std::cout << "Применяем способность: " << abilityManager.getAbilities().front()->getName() << std::endl;
            abilityManager.useAbility(field, manager, 2, 8);
        } else {
            std::cout << "Нет доступных способностей для использования." << std::endl;
        }

        //field.attack(2, 8, manager);
        field.printField();
        std::cout << std::endl;

        Ship ship(1, 9, 9, ShipOrientation::Horizontal);
        field.attack(9, 9, manager);
        field.attack(9, 9, manager);

        // Проверяем, уничтожен ли корабль и добавляем случайную способность при уничтожении
        // смерть корабля можно проверять многими способами, например
        // на каждой итерации игры считать количество кораблей на поле
        bool isShipDestroyed = 1;
        if (isShipDestroyed) {
            std::cout << "Корабль уничтожен! Добавляем новую случайную способность." << std::endl;
            abilityManager.addRandomAbility(1);
        }

        // Вывод количества доступных способностей после добавления
        std::cout << "Количество доступных способностей: " << abilityManager.getAbilityCount() << std::endl;
        field.printField();
        std::cout << std::endl;
```

```
    return 0;
}
```

Файл DoubleDamage.h

```
#include "DoubleDamage.h"

void DoubleDamage::setCoordinates(int x, int y) {
    x_ = x;
    y_ = y;
}

void DoubleDamage::useAbility(GameField& field, ShipManager& manager) {
    std::cout << "Использована способность: Двойной урон!" << std::endl;
    field.attack(x_, y_, manager);
    field.attack(x_, y_, manager);
}

std::string DoubleDamage::getName() const {
    return "Double Damage";
}
```

Файл GunBlaze.h

```
#include "GunBlaze.h"

void GunBlaze::useAbility(GameField& field, ShipManager& manager) {
    std::cout << "Использована способность: Обстрел!" << std::endl;
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> disX(0, field.getWidth() - 1);
    std::uniform_int_distribution<> disY(0, field.getHeight() - 1);

    int attempts = 0;
    while (attempts < 100) {
        int randomX = disX(gen);
        int randomY = disY(gen);
```

```
            if      (field.getCellStatus(randomX,      randomY)      ==
CellStatus::Ship) {
                field.attack(randomX, randomY, manager);
                std::cout  <<  "Урон  нанесен  по  сегменту  корабля  на
координатах (" << randomX << ", " << randomY << ")" << std::endl;
                return;
            }
            attempts++;
        }
        std::cout  <<  "Не  удалось  найти  корабль  для  атаки  после  100
попыток" << std::endl;
    }

    std::string GunBlaze::getName() const {
        return "Gun Blaze";
    }
```

Файл Scanner.h

```
    #include "Scanner.h"

    void Scanner::setCoordinates(int x, int y) {
        x_ = x;
        y_ = y;
    }

    void Scanner::useAbility(GameField& field, ShipManager& manager) {
        std::cout << "Использована способность: Сканер!" << std::endl;
        for (int dx = 0; dx <= 1; ++dx) {
            for (int dy = 0; dy <= 1; ++dy) {
                int currentX = x_ + dx;
                int currentY = y_ + dy;
                if    (field.isWithinBounds(currentX,    currentY)    &&
field.getCellStatus(currentX, currentY) == CellStatus::Ship) {
                    std::cout  <<  "Корабль  найден  на  клетке  ("  <<
currentX << ", " << currentY << ")" << std::endl;
                } else {
                    std::cout << "В клетке (" << currentX << ", " <<
currentY << ") ничего нет" << std::endl;
```

```
            }
        }
    }
}

std::string Scanner::getName() const {
    return "Scanner";
}
```

Результат работы программы:

```
mishalinux@LAPTOP-4DTT87P5:/mnt/c/users/misha/desktop/leti/oop$ ./main
0 0 0 0 0 0 0 0 0 0
0 2 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 2 2 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Использована способность: Двойной урон!
1 1 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 2 2 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Использована способность: Сканер!
В клетке (0, 8) ничего нет
В клетке (0, 9) ничего нет
Корабль найден на клетке (1, 8)
В клетке (1, 9) ничего нет
Использована способность: Обстрел!
Урон нанесен по сегменту корабля на координатах (2, 8)
1 1 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 2 2 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Применяем способность: Double Damage
Использована способность: Двойной урон!
1 1 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 2 1 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Корабль уничтожен! Добавляем новую случайную способность.
Добавлена новая случайная способность!
Количество доступных способностей: 3
1 1 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 2 1 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
```