**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №2**

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

# **Тема: Полиморфизм.**

|  |  |
| --- | --- |
| Студент гр. 3382 | Миллер С. Е. |
| Преподаватель | Жангиров Т. Р. |

Санкт-Петербург

2024

**Цель работы:**

Научиться работать с классами на языке программирования C++, в процессе разработать систему способностей для игры, включающую интерфейс способностей, реализацию различных типов способностей, менеджер способностей и обработку исключительных ситуаций.

**Задание:**

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

* Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
* Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
* Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

1. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
2. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
3. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

* Попытка применить способность, когда их нет
* Размещение корабля вплотную или на пересечении с другим кораблем
* 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

**Теоретические сведения:**

Классы в C++ представляют собой пользовательский тип данных, который объединяет данные и функции, работающие с этими данными. Они являются основой объектно-ориентированного программирования (ООП) и позволяют моделировать объекты.

Структура класса

Класс состоит из полей и методов. Поля хранят состояние объекта, а методы определяют его поведение. Обычно поля объявляются как private, чтобы защитить данные от прямого доступа извне. Методы могут быть как public, так и private, что позволяет контролировать доступ к функциональности класса.

Конструктор класса

Конструктором называют специальный метод, который вызывается при создании объекта и используется для инициализации его полей. Конструкторы могут принимать параметры, что позволяет создавать разные варианты инициализации.

Ключевые компоненты класса

* Поля хранят состояние объекта и обычно защищены от внешнего доступа.
* Методы определяют поведение объекта и могут быть открыты или закрыты для доступа.
* Конструкторы и деструкторы используются для создания и очистки объектов.
* Наследование позволяет создавать новые классы на основе существующих, что способствует повторному использованию кода.
* Инкапсуляция скрывает внутреннее состояние объекта и предоставляет доступ к нему через методы, что улучшает безопасность кода.
* Полиморфизм позволяет объектам разных классов обрабатывать вызовы методов по-разному, увеличивая гибкость программы.

Исключения в C++ представляют собой механизм обработки ошибок, позволяющий отделить код, вызывающий ошибку, от кода, который её обрабатывает. Когда возникает исключение, выполнение текущей функции прерывается, и управление передаётся блоку catch, соответствующему типу исключения. Это позволяет избежать распространения ошибок по коду, улучшая структуру и читаемость программы.

Рекомендуется использовать пользовательские типы исключений для более точной обработки ошибок. Иерархия классов для исключений позволяет эффективно обрабатывать различные типы ошибок, а переброс исключений - передавать управление обработкой на более высокий уровень.

Данные аспекты помогают эффективно использовать классы в C++, создавая более структурированные и поддерживаемые программы.

**Выполнение работы**

Класс Ability - интерфейс для способностей в игре. Он содержит виртуальный метод apply, который должен быть переопределен в производных классах для реализации конкретной логики применения способности. Также объявлен виртуальный деструктор, который делает класс Ability полиморфным и позволяет безопасно удалять объекты производных классов.

Класс Scanner реализует способность сканирования, наследуясь от базового класса Ability. В методе apply способность запрашивает у пользователя координаты x и y, а затем проверяет статус всех клеток в квадрате 2x2 вокруг этих координат. Если на какой-либо из этих клеток находится корабль, то сообщение о том, что корабль обнаружен, выводится на экран. В противном случае выводится сообщение о том, что сканирование не обнаружило кораблей.

Класс RandomShot реализует способность обстрела, наследуясь от базового класса Ability. В методе apply способность сначала проверяет, есть ли в игре хотя бы один не уничтоженный корабль. Если есть, то выбирается случайный корабль из списка кораблей, и этот выбор повторяется до тех пор, пока не будет найден не уничтоженный корабль. Затем выбирается случайный не уничтоженный сегмент этого корабля, и по нему производится выстрел, используя. В конце выводится сообщение о том, что случайный выстрел попал в корабль.

Класс DoubleDamage реализует способность "Двойной урон", наследуясь от базового класса Ability. В методе apply способность активирует режим двойного урона на игровом поле, устанавливает флаг, влияющий на расчет урона при последующих атаках. После этого выводится сообщение о том, что способность "Двойной урон" активирована.

Класс AbilityManager отвечает за управление способностями в игре, включая их создание, применение и добавление. В конструкторе класса вызывается метод generateInitialAbilities, который инициализирует начальный набор способностей. Этот метод создает массив с типами способностей и перемешивает его для случайного порядка. Затем, в зависимости от перемешанных типов, создаются объекты способностей, такие как DoubleDamage, Scan и RandomShot, которые добавляются в очередь способностей. Метод applyAbility отвечает за применение способности к игровому полю и менеджеру кораблей. Если в очереди есть способности, он извлекает первую из них, применяет ее к переданным объектам GameField и ShipManager, а затем удаляет использованную способность из памяти. Если очередь пуста, выбрасывается исключение NoAbilitiesException, указывающее на отсутствие доступных способностей. Метод addAbility позволяет добавлять новые способности в очередь, принимая указатель на объект Ability. Метод giveRandomAbility генерирует случайную способность и добавляет ее в очередь, используя генератор случайных чисел. Он также выводит сообщение о том, какая способность была добавлена. Этот класс обеспечивает гибкое управление способностями, позволяя как инициализировать их в начале игры, так и добавлять новые в процессе игры.

Класс GameException представляет собой базовый класс для исключений, специфичных для игры. Он наследуется от стандартного класса std::exception и переопределяет метод what(), который возвращает строку с описанием исключения, хранящегося в приватном поле message. Конструктор GameException(const std::string& message) принимает строку с описанием исключения в качестве аргумента и инициализирует поле message.

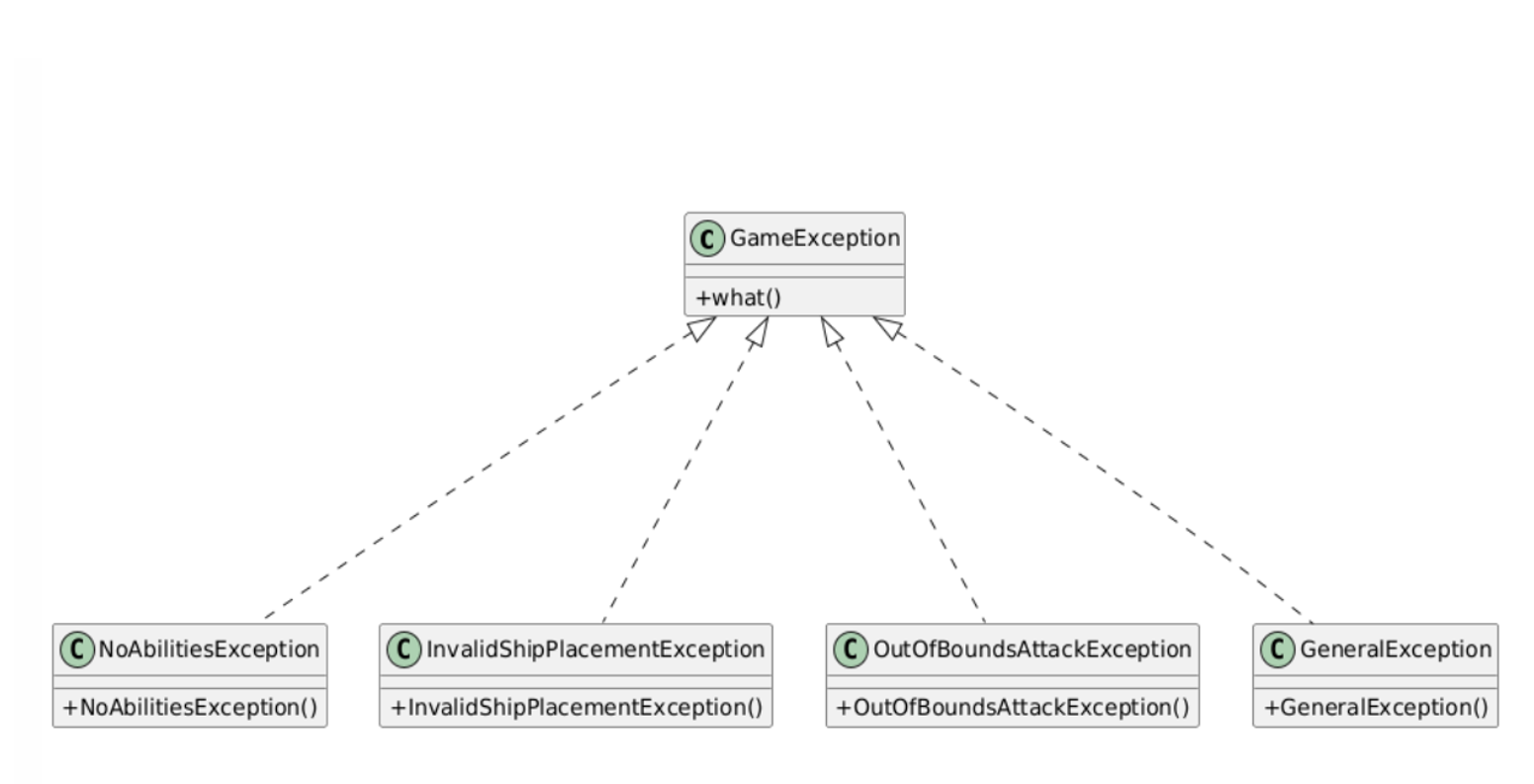
Класс NoAbilitiesException является производным от класса GameException и представляет собой специфическое исключение, которое возникает, когда игрок пытается применить способность, а в очереди способностей ничего нет. Конструктор NoAbilitiesException() вызывает конструктор базового класса GameException, передавая ему строку "No abilities available.", которая используется для описания этого исключения.

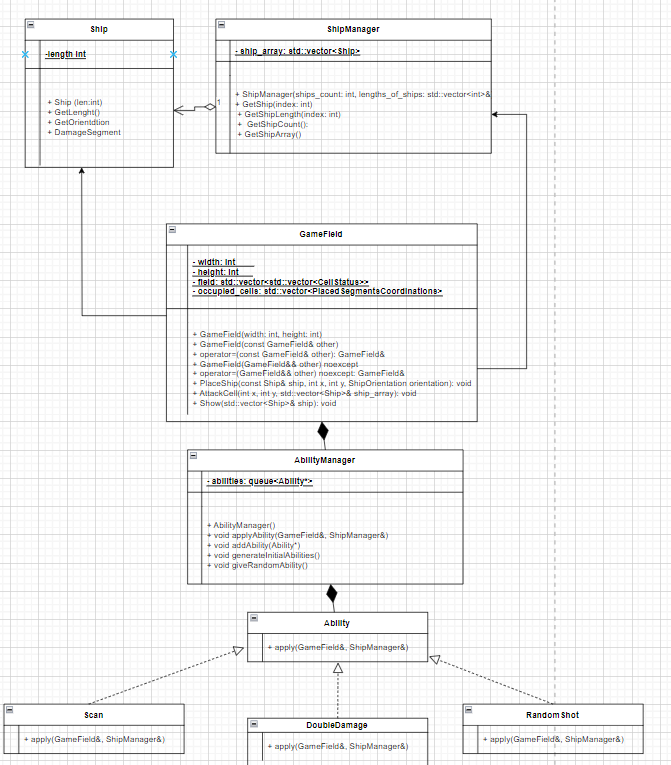
Класс InvalidShipPlacementException является производным от класса GameException и представляет собой специфическое исключение, которое возникает при попытке разместить корабль на игровом поле некорректно. Конструктор InvalidShipPlacementException(const std::string& reason) вызывает конструктор базового класса GameException, передавая ему строку "Invalid ship placement: " с дополнением reason, которая содержит более подробную информацию о причине ошибки размещения корабля.

Класс OutOfBoundsAttackException является производным от класса GameException и представляет собой специфическое исключение, которое возникает при попытке игрока атаковать клетку за пределами игрового поля. Конструктор OutOfBoundsAttackException() вызывает конструктор базового класса GameException, передавая ему строку "Attack coordinates out of bounds.", которая используется для описания этого исключения.

Класс GeneralException является производным от класса GameException и представляет собой специфическое исключение, которое используется для обработки непредусмотренных ошибок в игре. Конструктор GeneralException(const std::string& message) вызывает конструктор базового класса GameException, передавая ему строку message, которая содержит более подробную информацию об ошибке. Данный класс позволяет отловить непредусмотренные ошибки и выдать пользователю более информативное сообщение о произошедшем.

***UML - диаграмма:***



****

**Вывод**

В ходе работы была успешно разработана система способностей для игры, отвечающая поставленной цели. Реализован интерфейс способностей, позволяющий унифицировать использование различных типов способностей, таких как "Двойной урон", "Сканер" и "Обстрел". Создан менеджер способностей, который хранит очередь доступных способностей, управляет их применением и предоставляет механизм получения новой способности при уничтожении вражеского корабля. Для повышения надежности системы разработаны и реализованы классы-исключения для обработки нештатных ситуаций, таких как попытка применить способность при отсутствии доступных, размещение корабля вплотную к другому, а также атака за пределы игрового поля.

**Приложение 1**

*Файл main.cpp:*

#include <iostream>

#include <vector>

#include <string>

#include "GameField.h"

#include "ShipManager.h"

#include "AbilityManager.h"

int main()

{

// Пример использования классов

std::vector<int> ship\_lengths = {3, 2, 3, 2, 7};

ShipManager ship\_manager(5, ship\_lengths);

GameField my\_field(10, 10);

AbilityManager abilityManager;

// Размещение кораблей на поле

my\_field.PlaceShip(ship\_manager.GetShip(0), 1, 1, ShipOrientation::Vertical);

my\_field.PlaceShip(ship\_manager.GetShip(1), 3, 3, ShipOrientation::Horizontal);

my\_field.PlaceShip(ship\_manager.GetShip(2), 9, 8, ShipOrientation::Vertical);

my\_field.PlaceShip(ship\_manager.GetShip(3), 1, 2, ShipOrientation::Horizontal);

my\_field.PlaceShip(ship\_manager.GetShip(4), 0, 5, ShipOrientation::Horizontal);

// Пример атаки

if (my\_field.AttackCell(1, 1, ship\_manager.GetShipArray()))

{

abilityManager.giveRandomAbility();

}

if (my\_field.AttackCell(1, 1, ship\_manager.GetShipArray()))

{

abilityManager.giveRandomAbility();

}

if (my\_field.AttackCell(3, 3, ship\_manager.GetShipArray()))

{

abilityManager.giveRandomAbility();

}

my\_field.Show(ship\_manager.GetShipArray());

return 0;

}

*Файл Ship.h:*

#pragma once

#include <vector>

#include <stdexcept>

#include <string>

enum class CellStatus { Empty, Unknown, Ship };

enum class ShipOrientation { Horizontal, Vertical };

enum class SegmentStatus { Intact, Damaged, Destroyed };

class Ship {

public:

Ship(int length);

// Возвращает длину корабля

int GetLength() const;

// Возвращает ориентацию корабля

ShipOrientation GetOrientation() const;

// Возвращает статус сегмента по индексу

SegmentStatus GetSegmentStatus(int index) const;

// Наносит урон сегменту корабля по индексу

void DamageSegment(int index);

// Проверяет, уничтожен ли корабль

bool IsDestroyed() const;

int GetShipIndex() const;

void SetShipIndex(int index);

private:

int length\_;

int index;

ShipOrientation orientation\_;

std::vector<SegmentStatus> segments\_;

};

*Файл Ship.cpp:*

#include <iostream>

#include "Ship.h"

Ship::Ship(int length)

: length\_(length), orientation\_(ShipOrientation::Horizontal) {

if (length\_ > 4 ||length\_ < 1) {

std::cout << "Ship length must be between 1 and 4 inclusive." << '\n';

return;

}

segments\_.resize(length, SegmentStatus::Intact);

}

int Ship::GetLength() const { return length\_; }

ShipOrientation Ship::GetOrientation() const { return orientation\_; }

SegmentStatus Ship::GetSegmentStatus(int index) const {

return segments\_[index];

}

void Ship::DamageSegment(int index) {

if (index < 0 || index >= length\_) {

std::cout << "Invalid segment index." << '\n';

return;

}

if (segments\_[index] == SegmentStatus::Intact) {

segments\_[index] = SegmentStatus::Damaged;

}

else if (segments\_[index] == SegmentStatus::Damaged) {

segments\_[index] = SegmentStatus::Destroyed;

}

}

bool Ship::IsDestroyed() const {

for (auto status : segments\_) {

if (status != SegmentStatus::Destroyed) {

return false;

}

}

return true;

}

int Ship::GetShipIndex() const { return index; }

void Ship::SetShipIndex(int index) { this->index = index; }

*Файл ShipManager.h:*

#pragma once

#include <vector>

#include "Ship.h"

class ShipManager {

public:

ShipManager(int ships\_count, std::vector<int>& lengths\_of\_ships);

// Возвращает ссылку на корабль по индексу

Ship& GetShip(int index);

// Возвращает длину корабля по индексу

int GetShipLength(int index) const;

int GetShipCount() const;

std::vector<Ship>& GetShipArray();

bool AllShipsDestroy();

private:

std::vector<Ship> ship\_array;

};

*Файл ShipManager.cpp:*

#include <iostream>

#include "ShipManager.h"

ShipManager::ShipManager(int ships\_count, std::vector<int>& lengths\_of\_ships)

{

for (int i = 0; i < ships\_count; i++) {

if (lengths\_of\_ships[i] > 0 & lengths\_of\_ships[i] < 4){

ship\_array.emplace\_back(lengths\_of\_ships[i]);

ship\_array[i].SetShipIndex(i);

}

else{

std::cout << "Ship length must be between 1 and 4 inclusive." << '\n';

return;

}

}

}

Ship& ShipManager::GetShip(int index)

{

return ship\_array[index];

}

// Возвращает длину корабля по индексу

int ShipManager::GetShipLength(int index) const {

if (index < 0 || index >= (int) ship\_array.size()) {

std::cout<< "Invalid ship index.";

return 0;

}

return ship\_array[index].GetLength();

}

int ShipManager::GetShipCount() const

{

return ship\_array.size();

}

std::vector<Ship>& ShipManager::GetShipArray()

{

return ship\_array;

}

bool ShipManager::AllShipsDestroy()

{

for(auto& ship:ship\_array){

if (!ship.IsDestroyed()){

return false;

}

}

}

*Файл GameField.h:*

#pragma once

#include <vector>

#include <stdexcept>

#include "Ship.h"

#include "ShipManager.h"

struct PlacedSegmentsCoordinations

{

int x;

int y;

int ship\_index;

int segment\_index;

};

class GameField {

public:

GameField(int width, int height);

GameField(const GameField& other);

GameField& operator = (const GameField& other);

GameField(GameField&& other) noexcept;

GameField& operator = (GameField&& other) noexcept;

void PlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation);

bool AttackCell(int x, int y, std::vector<Ship> &ship\_array);

void Show(std::vector<Ship> &ship);

void setDoubleDamage(bool enable);

int getWidth () const { return width; }

int getHeight () const { return height; }

CellStatus getCellStatus(int x, int y) { return field[x][y]; }

private:

int width;

int height;

std::vector<std::vector<CellStatus>> field;

std::vector<PlacedSegmentsCoordinations> occupied\_cells;

bool double\_damage;

};

*Файл GameField.cpp:*

#include "GameField.h"

#include "OutOfBoundsAttackException.h"

#include "InvalidShipPlacementException.h"

#include <iostream>

GameField::GameField(int width, int height) : width(width), height(height) {

if (width < 1 || height < 1) {

std::cout << "Field width and height must be greater than 0.";

return;

}

field.resize(height, std::vector<CellStatus>(width, CellStatus::Unknown));

}

//конструктор копирования

GameField::GameField(const GameField &other)

{

width = other.width;

height = other.height;

field = other.field;

occupied\_cells = other.occupied\_cells;

}

//оперотор присваивания копирования

GameField& GameField::operator=(const GameField& other)

{

if (this != &other)

{

width = other.width;

height = other.height;

field = other.field;

occupied\_cells = other.occupied\_cells;

}

return \*this;

}

// Конструктор перемещения

GameField::GameField(GameField &&other) noexcept

{

std::swap(width, other.width);

std::swap(height, other.height);

field = std::move(other.field);

occupied\_cells = std::move(other.occupied\_cells);

}

// Оператор присваивания перемещением

GameField& GameField::operator=(GameField&& other) noexcept

{

if (this != &other)

{

std::swap(width, other.width);

std::swap(height, other.height);

field = std::move(other.field);

occupied\_cells = std::move(other.occupied\_cells);

}

return \*this;

}

void GameField::PlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation)

{

//проверка на выход за границы

if (x < 0 || y >= height || x >= width || y < 0

|| (orientation == ShipOrientation::Horizontal && x + ship.GetLength() >= width)

|| (orientation == ShipOrientation::Vertical && y + ship.GetLength() >= height))

{

throw OutOfBoundsAttackException();

}

//проерка на пересечение или соприкосновение кораблей

for (int i = 0; i < ship.GetLength(); i++){

if (orientation == ShipOrientation::Horizontal){

if(field[y][x + i] == CellStatus::Ship

|| (y + 1 < height && field[y + 1][x + i] == CellStatus::Ship)

|| (y - 1 >= 0 && field[y - 1][x + i] == CellStatus::Ship))

{

throw InvalidShipPlacementException(" Ship intersects or touches with other ships '\n'");

return;

}

}

else{

if (field[y + i][x] == CellStatus::Ship

|| (x + 1 < width && field[y + i][x + 1] == CellStatus::Ship)

|| (x - 1 >= 0 && field[y + i][x - 1] == CellStatus::Ship))

{

throw InvalidShipPlacementException("Ship intersects or touches with other ships '\n'");

return;

}

}

}

for (int i = 0; i < ship.GetLength(); i++)

{

if (orientation == ShipOrientation::Horizontal)

{

occupied\_cells.emplace\_back(PlacedSegmentsCoordinations{.x = x + i, .y = y, .ship\_index = ship.GetShipIndex(), .segment\_index = i});

field[y][x + i] = CellStatus::Ship;

}

else

{

occupied\_cells.emplace\_back(PlacedSegmentsCoordinations{.x = x, .y = y + i, .ship\_index = ship.GetShipIndex(), .segment\_index = i});

field[y + i][x] = CellStatus::Ship;

}

}

}

bool GameField::AttackCell(int x, int y, std::vector<Ship> &ship\_array)

{

if (x >= width || x < 0 || y >= height || y < 0)

{

throw OutOfBoundsAttackException();

}

int ship\_ind = -1;

for (auto &cell : occupied\_cells)

{

if (cell.x == x && cell.y == y)

{

ship\_array[ship\_ind].DamageSegment(cell.segment\_index);

if(double\_damage){

ship\_array[ship\_ind].DamageSegment(cell.segment\_index);

double\_damage = false;

}

if (ship\_array[ship\_ind].IsDestroyed()){

return true;

}

}

}

field[x][y] = CellStatus::Empty;

return false;

}

*Файл AbilityManager.h:*

#pragma once

#include <vector>

#include <queue>

#include <random>

#include "ability/Ability.h"

#include "ability/DoubleDamage.h"

#include "ability/Scan.h"

#include "ability/RandomShot.h"

#include "NoAbilitiesException.h"

class AbilityManager {

public:

AbilityManager();

void applyAbility(GameField& field, ShipManager& manager);

void addAbility(Ability\* ability);

void generateInitialAbilities();

void giveRandomAbility();

private:

std::queue<Ability\*> abilities;

};

*Файл AbilityManager.cpp:*

#include "AbilityManager.h"

AbilityManager::AbilityManager() {

generateInitialAbilities();

}

void AbilityManager::applyAbility(GameField& field, ShipManager& manager) {

if (!abilities.empty()) {

Ability\* ability = abilities.front();

abilities.pop();

ability->apply(field, manager);

delete ability;

}

else {

throw NoAbilitiesException();

}

}

void AbilityManager::addAbility(Ability\* ability) {

abilities.push(ability);

}

void AbilityManager::generateInitialAbilities() {

srand(time(NULL)); // Инициализация генератора случайных чисел

// Массив с типами способностей (0 - DoubleDamage, 1 - Scanner, 2 - RandomShot)

int abilityTypes[3] = { 0, 1, 2 };

// Перемешивание массива для случайного порядка способностей

for (int i = 2; i > 0; --i) {

int j = rand() % (i + 1);

std::swap(abilityTypes[i], abilityTypes[j]);

}

// Создание и добавление способностей в очередь

for (int i = 0; i < 3; ++i) {

switch (abilityTypes[i]) {

case 0:

addAbility(new DoubleDamage());

break;

case 1:

addAbility(new Scan());

break;

case 2:

addAbility(new RandomShot());

break;

}

}

}

void AbilityManager::giveRandomAbility() {

srand(time(NULL)); // Инициализация генератора случайных чисел

int abilityType = rand() % 3; // Случайное число от 0 до 2

switch (abilityType) {

case 0:

std::cout << "\nAdded double damage ability!\n\n";

addAbility(new DoubleDamage());

break;

case 1:

std::cout << "\nAdded scanner ability!\n\n";

addAbility(new Scan());

break;

case 2:

std::cout << "\nAdded RandomShot ability!\n\n";

addAbility(new RandomShot());

break;

}

}

*Файл Ability.h:*

#pragma once

#include "GameField.h"

#include "ShipManager.h"

class Ability {

public:

virtual void apply(GameField& field, ShipManager& manager) = 0;

virtual ~Ability() = default;

};

*Файл RandomShot.h:*

#pragma once

#include "Ability.h"

#include <random>

class RandomShot : public Ability {

public:

void apply(GameField& field, ShipManager& manager) override;

};

*Файл RandomShot.cpp:*

#include "RandomShot.h"

#include <random>

void RandomShot::apply(GameField& field, ShipManager& manager) {

srand(time(NULL));

if (manager.GetShipCount() > 0) {

Ship\* ship = nullptr;

int random\_index;

do {

random\_index = rand() % manager.GetShipCount();

ship = &manager.GetShip(random\_index);

} while (ship->IsDestroyed());

int segment\_index;

do {

segment\_index = rand() % ship->GetLength();

} while (ship->GetSegmentStatus(segment\_index) != SegmentStatus::Destroyed);

ship->DamageSegment(segment\_index);

std::cout << "\nRandom shot hit the ship!\n\n";

}

}

*Файл Scan.h:*

#pragma once

#include "Ability.h"

class Scanner : public Ability {

public:

void apply(GameField& field, ShipManager& manager) override;

};

*Файл Scan.cpp:*

#include "Scan.h"

void Scan::apply(GameField& field, ShipManager& manager) {

int x, y;

bool flag = true;

std::cout << "Scanner activated! (x y): ";

std::cin >> x >> y;

for (int i = -1; i <= 1; ++i) {

for (int j = -1; j <= 1; ++j) {

int scanX = x + i;

int scanY = y + j;

if (scanX >= 0 && scanX < field.getWidth() && scanY >= 0 && scanY < field.getHeight()) {

if (field.getCellStatus(scanX, scanY) == CellStatus::Ship) {

std::cout << "Ship detected at (" << scanX << ", " << scanY << ")\n";

flag = false;

}

}

}

if (flag) {

std::cout << "Scanner didn't see the ships\n";

}

}

*Файл DoubleDamage.h:*

#pragma once

#include "Ability.h"

class DoubleDamage : public Ability {

public:

void apply(GameField& field, ShipManager& manager) override;

};

*Файл DoubleDamage.cpp:*

#include "DoubleDamage.h"

void DoubleDamage::apply(GameField& field, ShipManager& manager) {

field.setDoubleDamage(true);

std::cout << "Ability 'Double Damage' activated!\n";

}

*Файл GameException.h:*

#pragma once

#include <string>

#include <stdexcept>

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;

};

*Файл NoAbilitiesException.h:*

#pragma once

#include "GameException.h"

class NoAbilitiesException : public GameException {

public:

NoAbilitiesException() : GameException("No abilities available.") {}

};

*Файл OutOfBoundsAttackException.h:*

#pragma once

#include "GameException.h"

class OutOfBoundsAttackException : public GameException {

public:

OutOfBoundsAttackException() : GameException("Attack coordinates out of bounds.") {}

};

*Файл InvalidShipPlacementException.h:*

#pragma once

#include "GameException.h"

class InvalidShipPlacementException : public GameException {

public:

InvalidShipPlacementException(const std::string& reason) : GameException("Invalid ship placement: " + reason) {}

};

*Файл GeneralException.h:*

#pragma once

#include "GameException.h"

class GeneralException : public GameException {

public:

GeneralException(const std::string& message) : GameException(message) {}

};