**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе № 2**

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

Тема: «Полиморфизм»



|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

# Цель работы

Изучить основы полиморфизма классов в ООП C++, создать класс-интерфейс способности и три способности на его основе, класс менеджера способностей, реализовать набор классов-исключений для различных ситуаций.

# Задание

1. Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:
2. Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
3. Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
4. Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.
5. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
6. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
7. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):
8. Попытка применить способность, когда их нет
9. Размещение корабля вплотную или на пересечении с другим кораблем
10. 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

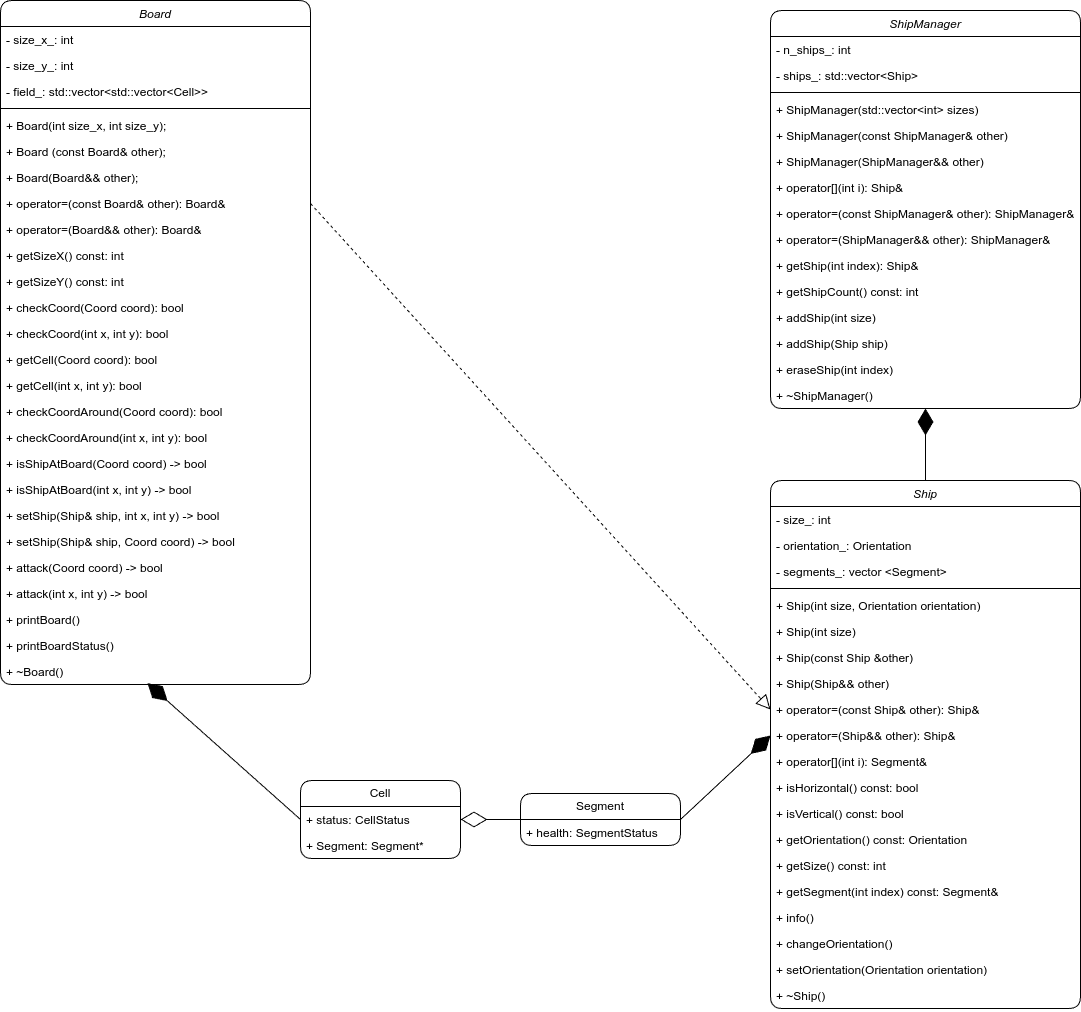


Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Ship*, *ShipManager*, Board и необходимых структур для функционирования программы *Segment*, *Cell*.

Классы *Ship*, *ShipManager* и *Board* реализованы по условию, их описание и методы присутствуют ниже. Структура *Segment* нужна для связи здоровья сегмента корабля с его координатами. Структура *Cell* нужна для хранения информации о конкретной ячейке поля (известно ли её состояние или нет) и сегмента корабля (если он там присутствует) в ней. Благодаря структурам возможно создать косвенную связь между полем и кораблями (*Board* – *Cell* – *Segment* | *ShipManager* – *Ship* – *Segment*).

Класс *Board* связан со структурой *Cell* с помощью композиции, потому что является частью поля , также оно управляет существованием ячейки. Аналогичная связь между *ShipManager* и *Ship,* *Ship* и *Segment*.

Структура *Cell* связана со структурой *Segment* связаны агрегацией (имеет), потому что в ячейке есть ссылка на сегмент, но она не является владельцем сегментом и не управляет временем её жизни.

Классы *Board* и *Ship* связаны с помощью зависимости, потому что *Board* принимает *Ship* как аргумент в функции расположения корабля, и, реализация *Board* зависит от *Ship*.

*Ship* является классом корабля. Он имеет следующие поля:

* *int* size\_ – длина корабля.
* *Orientation orientation\_* – ориентация корабля на поле (горизонтальная/вертикальная).
* *vector<Segment> segments\_* – сегменты корабля.

И следующие методы:

* *Ship(int size);* Ship::Ship(int size, int orientation); – конструкторы класса, создающий корабль.
* *~Ship()* – деструктор класса, освобождающий память сегментов.
* *int getSize() const* – возвращает длину корабля.
* *Segment\* getSegment() const* – возвращает сегмент корабля.
* *bool isHorizontal() const* – проверяет, что у корабля горизонтальная ориентация
* *bool isVertical() const* – проверяет, что у корабля вертикальная ориентация
* *void changeOrientation()* – изменяет ориентацию корабля.
* *void info() const* – выводит информацию о корабле и его сегментах.

Класс *ShipManager* отвечает за корабли, вызов и проверка различных функций взаимодействия, он хранит информацию о них. Он имеет следующие поля:

* *vector<Ship>ships* – возвращает все привязанные корабли.

И следующие методы:

* *ShipManager(std::vector<int> sizes) –* конструктор класса, создающий менеджер кораблей с кораблями, указанными в векторе sizes.
* *ShipManager(const ShipManager& other) –* конструктор копирования, создающий копию другого менеджера кораблей.
* *ShipManager(ShipManager&& other) –* конструктор перемещения, перемещающий содержимое другого менеджера кораблей.
* *Ship& operator[](int index) –* оператор индексации, возвращающий ссылку на корабль по указанному индексу.
* *ShipManager& operator=(const ShipManager& other) –* оператор присваивания копированием, копирующий содержимое другого менеджера кораблей.
* *ShipManager& operator=(ShipManager&& other) –* оператор присваивания перемещением, перемещающий содержимое другого менеджера кораблей.
* *Ship& getShip(int index) –* возвращает ссылку на корабль по указанному индексу.
* *int getShipCount() const –* возвращает количество кораблей в менеджере.
* *void addShip(int size) –* добавляет новый корабль заданного размера в менеджер.
* *void addShip(Ship ship) –* добавляет существующий корабль в менеджер.
* *void eraseShip(int index) –* удаляет корабль по указанному индексу из менеджера.
* *~ShipManager() –* деструктор класса, освобождающий память, занятую кораблями.

Класс *Board* отвечает за работу поля: размещение кораблей, изменение состояния ячеек. Он имеет следующие поля:

* *int size\_y\_* – количество рядов (размер поля по y).
* *int* size\_x\_ – количество столбцов (размер поля по x).
* *vector<Cell> Board* – поле, состоящее из ячеек.

И следующие методы:

* *Board(int size\_x, int size\_y) –* конструктор класса, создающий игровое поле с заданными размерами.
* *Board(const Board& other) –* конструктор копирования, создающий копию другого игрового поля.
* *Board(Board&& other) –* конструктор перемещения, перемещающий содержимое другого игрового поля.
* *Board& operator=(const Board& other) –* оператор присваивания копированием, копирующий содержимое другого игрового поля.
* *Board& operator=(Board&& other) –* оператор присваивания перемещением, перемещающий содержимое другого игрового поля.
* *int getSizeX() const –* возвращает ширину поля.
* *int getSizeY() const –* возвращает высоту поля.
* *bool checkCoord(Coord coord) const –* проверяет, находится ли координата coord в пределах поля.
* *bool checkCoord(int x, int y) const –* проверяет, находится ли координата (x, y) в пределах поля.
* *Cell& getCell(Coord coord) –* возвращает ссылку на ячейку по координатам coord.
* *Cell& getCell(int x, int y) –* возвращает ссылку на ячейку по координатам (x, y).
* *bool checkCoordAround(Coord coord) –* проверяет, находятся ли все соседние ячейки вокруг координаты coord в пределах поля.
* *bool checkCoordAround(int x, int y) –* проверяет, находятся ли все соседние ячейки вокруг координаты (x, y) в пределах поля.
* *bool isShipAtBoard(Coord coord) –* проверяет, находится ли корабль в ячейке с координатами coord.
* *bool isShipAtBoard(int x, int y) –* проверяет, находится ли корабль в ячейке с координатами (x, y).
* *bool setShip(Ship& ship, Coord coord) –* размещает корабль ship на поле, начиная с координаты coord. Возвращает true, если размещение успешно.
* *bool attack(Coord coord) –* атакует ячейку с координатами coord. Возвращает true, если атака успешна.
* *bool attack(int x, int y) – атакует ячейку с координатами (x, y).* Возвращает true, если атака успешна.
* *bool setShip(Ship& ship, int x, int y) –* размещает корабль ship на поле, начиная с координат (x, y). Возвращает true, если размещение успешно.
* *void printBoard() –* выводит текущее состояние игрового поля.
* *void printBoardStatus() –* выводит статус игрового поля (например, количество живых кораблей).
* *~Board() –* деструктор класса, освобождающий память, занятую полем.

Структура *Segment* отвечает за хранение информации о местоположении сегмента корабля и его здоровье. Она имеет следующие поля:

* *SegmentStatus health* – здоровье поля (kWhole/kDamaged/kDestroyed).

Структура *Cell* отвечает за хранение информации о ячейке поля. Она имеет следующие поля:

* *Coordinate coordinate* – координаты ячейки.
* *CellState state* – открыто поле или нет (Hidden/Revealed).
* *CellValue value* – значение ячейки поля.
* *Segment\* segment = nullptr* – указатель на сегмент корабля в этой ячейке.

# Тестирование:

В

# Выводы

Во время выполнения лабораторной работы были изучены основы объектно-ориентированного программирования, реализованы классы на языке С++, написан первый прототип программы для игры в морской бой.

Во время выполнения лабораторной работы были изучены основы полиморфизма классов в ООП C++, был класс-интерфейс способности и три способности на его основе: двойной урон, сканер и обстрел, класс менеджера способностей для использования данной способности, реализован набор классов-исключений для различных ситуаций.