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. 3344 _____ Коршунов П.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Освоение основ объектно-ориентированного программирования путем создания классов, взаимодействующих между собой для реализации игровой логики.

Задание.

А) Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.

Б) Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.

В) Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

1)неизвестно (изначально вражеское поле полностью неизвестно),

2)пустая (если на клетке ничего нет)

3)корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

Примечания:

Не забывайте для полей и методов определять модификаторы доступа

Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum

Не используйте глобальные переменные

При реализации копирования нужно выполнять глубокое копирование

При реализации перемещения, не должно быть лишнего копирования

При выделении памяти делайте проверку на переданные значения

У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

Выполнение работы.

Класс *Ship*

Класс ***Ship*** предназначен для представления корабля, который может быть размещен на игровом поле. Он содержит информацию о состоянии корабля и обеспечивает методы для управления его состоянием.

Поля класса

size: Целое число, определяющее размер корабля. Размер устанавливается при создании корабля и больше не изменяется. Размер варьируется от 1 до 4 и определяет количество сегментов, которые составляют корабль.

segments: Вектор перечислений ***SegmentStatus***, представляющий состояние каждого сегмента корабля. Вектор ***segments*** позволяет удобно хранить и отслеживать состояние каждого сегмента корабля, что упрощает работу с кораблем при нанесении ему урона.

orientation: Перечисление ***Orientation***, определяющее ориентацию корабля (вертикально или горизонтально). Это поле необходимо, чтобы понимать, как корабль размещен на поле.

Методы класса

Конструктор ***Ship(int size):*** Принимает размер корабля, инициализирует поле ***size*** и создает вектор ***segments*** с соответствующим количеством элементов, каждый из которых имеет начальное значение ***SegmentStatus::intact***. Такое решение позволяет установить изначальное состояние каждого сегмента, что логично, так как корабль в начале игры всегда целый.

int getSize() const: Возвращает размер корабля. Этот метод полезен для получения информации о количестве сегментов при взаимодействии с другими объектами, например при размещении корабля на поле.

Orientation getOrientation() const: Возвращает ориентацию корабля. Этот метод позволяет другим классам узнать, как корабль был размещен,

что необходимо при проверке расположения или при определении атакованных сегментов.

void setOrientation(Orientation orientation): Устанавливает ориентацию корабля. Метод позволяет задать ориентацию после создания объекта, обеспечивая гибкость в его размещении на игровом поле.

void damageSegment(int index): Наносит урон сегменту корабля по заданному индексу. Если сегмент целый, он становится поврежденным, если поврежденный — уничтожается. Такой метод обеспечивает реалистичное отслеживание состояния корабля при атаках на него.

Метод ***damageSegment*** обеспечивает возможность обновления состояния корабля при атаках.

Класс *ShipManager*

Класс ***ShipManager*** предназначен для управления всеми кораблями, принадлежащими игроку или противнику. Он отвечает за отслеживание всех кораблей одной стороны, их расположение на поле, нанесение урона и доступ к информации о кораблях.

Поля класса

struct hashPair: Это структура, реализующая оператор (), который позволяет вычислять хеш для пар значений типа ***std::pair<int, int>***. Структура содержит:
size_t operator()(const std::pair<T1, T2>& p) const: Метод, принимающий пару в качестве аргумента и возвращающий хеш этой пары.

shipsAmount: Целое число, представляющее общее количество кораблей, которыми управляет менеджер. Это значение необходимо для отслеживания количества кораблей и их обработки.

ships: Вектор объектов ***Ship***, содержащий все корабли, которыми управляет менеджер. Это основное хранилище данных о каждом корабле, включая его состояние и ориентацию. Использование вектора позволяет эффективно управлять кораблями и обращаться к ним по индексу.

shipCoordinatesMap: Хеш-таблица, в которой ключом является пара координат (x, y) на игровом поле, а значением — индекс корабля в векторе

ships. Эта структура данных обеспечивает быстрый доступ к информации о том, какой корабль занимает определенные координаты на поле. Причина использования этой структуры заключается в том, что она позволяет быстро находить, к какому кораблю относится конкретная координата на поле.

Методы класса

Конструктор ***ShipManager(int shipsAmount, std::initializer_list<int> shipsSizes):*** Инициализирует объект ***ShipManager*** заданным количеством кораблей и их размерами. В конструкторе происходит создание каждого корабля на основе переданных размеров и добавление его в вектор ***ships***. Если количество кораблей не совпадает с количеством размеров, генерируется исключение ***std::invalid_argument***. Такая проверка обеспечивает корректную инициализацию.

Ship& getShip(int index): Возвращает ссылку на корабль по заданному индексу. Используется для доступа к конкретному кораблю при его размещении или нанесении урона. В методе вызывается ***validateShipIndex***, чтобы избежать некорректного доступа к вектору.

Ship::SegmentHealth getShipSegmentStatus(int shipIndex, int segmentIndex): Возвращает здоровье определенного сегмента корабля. Используется для корректного отображения поля.

std::pair<int, int> getShipStartCoordinates(int x, int y): Возвращает начальные координаты корабля, которому принадлежат переданные координаты (x, y). Этот метод нужен для корректного нанесения урона, когда нужно найти определенный сегмент корабля.

void attackShip(int shipIndex, int segmentIndex): Наносит урон по кораблю с заданным индексом ***shipIndex*** в его сегменте с индексом ***segmentIndex***. Метод вызывает соответствующий метод у корабля ***damageSegment***.

void addShipCoordinates(int index, int x, int y): Добавляет в ***shipCoordinatesMap*** координаты (x, y) и связывает их с индексом index корабля. Этот метод вызывается при размещении корабля на игровом поле и позволяет в

дальнейшем быстро определять, к какому кораблю принадлежат заданные координаты.

int getShipIndexByCoordinates(int x, int y) const: Возвращает индекс корабля, расположенного в координатах (x, y). Если в этих координатах корабля нет, возвращает -1.

int getShipAmount() const: Возвращает количество кораблей. Метод используется для получения общей информации о менеджере.

void validateShipIndex(int index) const: Приватный метод, проверяющий, что индекс корабля находится в допустимых пределах. Вызывает ***std::out_of_range***, если индекс некорректен. Такой метод гарантирует, что остальные методы класса всегда работают с корректными данными.

Класс *GameField*

Класс ***GameField*** представляет игровое поле, на котором размещаются корабли, и управляет взаимодействием между игроком и вражескими кораблями. Он отвечает за размещение кораблей, проверку их статуса и обработку атак.

Поля класса

int width: Ширина поля.

int height: Высота поля.

std::vector<std::vector<CellStatus>> field: Двумерный вектор, представляющий состояние игрового поля для самого игрока. Содержит статусы клеток (неизвестно, пустая, занятая кораблем).

std::vector<std::vector<CellStatus>> enemyField: Двумерный вектор для представления вражеского поля. Хранит статус клеток, которые игрок открывает во время игры.

ShipManager* shipManager: Указатель на менеджер собственных кораблей. Позволяет управлять размещением и статусом кораблей игрока.

ShipManager* enemyShipManager: Указатель на менеджер вражеских кораблей. Позволяет отслеживать и изменять здоровье и статус кораблей противника.

Методы класса

Конструктор: ***GameField(int width, int height)*** - Инициализирует поля, проверяя корректность размеров. Создает два поля: одно для игрока и одно для противника с начальными статусами.

Копирующий конструктор: ***GameField(const GameField &other)*** - Обеспечивает глубокое копирование для правильного управления ресурсами.

Конструктор перемещения**:** ***GameField(GameField &&other) noexcept*** - Обеспечивает перемещение ресурсов.

Оператор присваивания:

GameField &operator=(const GameField &other): Глубокое копирование, чтобы изменения в одном экземпляре не влияли на другой.

GameField &operator=(GameField &&other) noexcept: Перемещение ресурсов, предотвращая ненужное копирование.

Методы управления менеджерами:

setShipManager(ShipManager* manager): Устанавливает менеджер для собственных кораблей.

setEnemyShipManager(ShipManager* manager): Устанавливает менеджер для вражеских кораблей. Оба метода обеспечивают возможность связи между полем и менеджерами, позволяя полю управлять состоянием кораблей.

Методы проверки и валидации:

bool isWithinBounds(int x, int y) const: Проверяет, находятся ли координаты в пределах игрового поля.

bool isCellOccupied(int x, int y) const: Определяет, занята ли клетка кораблем.

void validatePlacement(int x, int y, int size, Ship::Orientation orientation) const: Проверяет возможность размещения корабля, включая проверку на соприкосновение с другими кораблями.

Методы взаимодействия с полем:

void placeShip(int shipIndex, int x, int y, Ship::Orientation orientation): Размещает корабль на поле, вызывает валидацию перед размещением.

void attackCell(int x, int y): Обрабатывает атаку на клетку, используя координаты для определения, был ли нанесен удар по кораблю.

void printField(bool isEnemy) const: Выводит текущее состояние игрового поля, что полезно для отладки и визуализации.

Два поля: Наличие двух полей (своего и вражеского) позволяет разделить логику управления собственными кораблями и взаимодействия с вражескими.

Два менеджера: Использование отдельных менеджеров для собственных и вражеских кораблей позволяет точно контролировать их статус. ***enemyShipManager*** позволяет обрабатывать урон по вражеским кораблям, не раскрывая их положение на поле.

Тестирование

```
GameField field(10, 10);
GameField field1(10, 10);
field.printField();
std::cout << '\n';
field.printField(true);

ShipManager manager(3, {2, 3, 4});
ShipManager manager1(3, {2, 3, 4});

field.setShipManager(&manager);
field.setEnemyShipManager(&manager1);

field1.setShipManager(&manager1);
field1.setEnemyShipManager(&manager);

field.placeShip(0, 0, 0, Ship::Orientation::HORIZONTAL);
field.placeShip(1, 3, 3, Ship::Orientation::VERTICAL);
field.placeShip(2, 6, 9, Ship::Orientation::HORIZONTAL);

field1.placeShip(0, 3, 2, Ship::Orientation::HORIZONTAL);
field1.placeShip(1, 0, 2, Ship::Orientation::VERTICAL);
field1.placeShip(2, 5, 8, Ship::Orientation::HORIZONTAL);
```

Создание полей, создание менеджеров, установка менеджеров для полей, расстановка кораблей. Вывод полей до расстановки кораблей.

Атака полей по кораблям и в пустую клетку. Вывод полей и состояния сегментов корабля, по которому был нанесен урон.

```
field.attackCell(0, 0);
field.attackCell(6, 8);
std::cout << '\n';

field1.attackCell(0, 0);
std::cout << '\n';

field.printField(true);
std::cout << '\n';
field.printField();
std::cout << '\n';
for (Ship::SegmentHealth seg : manager1.getShip(2).getSegments())
{
    std::cout << static_cast<int>(seg);
}
std::cout << '\n';
```

```
Miss at (0, 0)
Hit at (6, 8)

Hit at (0, 0)

. ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? + ? ? ?
? ? ? ? ? ? ? ? ? ?

+ S . . . . . .
. . . . . .
. . . . . .
. . . S . . . . .
. . . S . . . . .
. . . S . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . S S S S

2122
```

Выводы.

Реализован класс для представления корабля с возможностью управления его состоянием (целый, поврежден, уничтожен).

Создан менеджер, который управляет коллекцией кораблей, обеспечивая методы для их доступа и обработки урона. Использование хеш-таблицы для хранения координат кораблей позволяет эффективно находить корабли по их позициям.

Разработано поле для игры, которое хранит информацию о своих и вражеских кораблях. Реализованы методы для проверки доступности клеток, размещения кораблей и обработки атак, что позволяет поддерживать игровые механики.

Копирование и перемещение объектов: Реализованы конструкторы копирования и перемещения, а также операторы присваивания для поля, что обеспечивает безопасное управление памятью и минимизацию затрат при работе с объектами.

UML-диаграмма реализованных классов.