**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**Тема:** **Связывание классов**

|  |  |
| --- | --- |
| Студент гр. 3344 | Анахин Е.Д. |
| Преподаватель | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Создать класс игры с игровым циклом, управляемым состоянием, реализовать сохранение и загрузку игры, используя идиому RAII для работы с файлами.

**Задание**.

1. Создать класс игры, который реализует следующий игровой цикл:
2. Начало игры
3. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
4. В случае проигрыша пользователь начинает новую игру
5. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.
6. 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.
7. 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.
8. Примечание:
9. Класс игры может знать о игровых сущностях, но не наоборот
10. Игровые сущности не должны сами порождать объекты состояния
11. Для управления самое игрой можно использовать обертки над командами
12. При работе с файлом используйте идиому RAII.

## Выполнение работы.

Был создан класс GameManager, который управляет жизненным циклом игры:

* GameManager()

Описание: Конструктор по умолчанию. Инициализирует игру, но основная настройка происходит в startGame.

* void startGame()

Описание: Запускает игру. Предлагает пользователю загрузить сохранение или начать новую игру. Затем запускает основной игровой цикл.

* void saveGame(const std::string& filename) const

Описание: Сохраняет текущее состояние игры в файл.

* void loadGame(const std::string& filename)

Описание: Загружает состояние игры из файла.

Также был создан класс Game, который выполняет все действия, которые происходят в игре.

**Game()**

* **Описание:** Конструктор по умолчанию. Инициализирует игру, создавая поля, менеджеры кораблей и способностей, а также расставляет корабли.

**void startNewGame()**

* **Описание:** Запускает новую игру, сбрасывая все состояния игры и инициализируя начальные условия.

**void startNewRound()**

* **Описание:** Запускает новый раунд игры, сбрасывая поле противника и расставляя его корабли.

**bool makeUserTurn(int x, int y)**

* **Описание:** Обрабатывает ход игрока, атакуя указанную клетку на поле противника. Возвращает true, если атака успешна, и false в противном случае.

**void makeEnemyTurn()**

* **Описание:** Обрабатывает ход противника, атакуя случайным образом клетку на поле игрока.

**bool isGameOver() const**

* **Описание:** Проверяет, закончена ли игра (все корабли одной из сторон уничтожены).

**bool hasUserWon() const**

* **Описание:** Проверяет, выиграл ли игрок (все корабли противника уничтожены).

**void useAbility()**

* **Описание:** Использует случайную способность из менеджера способностей.

**bool canUseAbility(int abilityIndex) const**

* **Описание:** Проверяет, можно ли использовать способность по указанному индексу.

**void printGameState() const**

* **Описание:** Выводит текущее состояние игры, включая поля игрока и противника, а также доступные способности.

**const Field& getUserField() const**

* **Описание:** Возвращает ссылку на поле игрока.

**const Field& getEnemyField() const**

* **Описание:** Возвращает ссылку на поле противника.

**int getCurrentRound() const**

* **Описание:** Возвращает текущий раунд игры.

Для сохранения и загрузки игры был также создан класс GameState, в котором были переопределены операции «>>» и «<<»

**GameState()**

* **Описание:** Конструктор по умолчанию. Инициализирует объект GameState без параметров.

**GameState(int width, int height, int round, const Field& userField, const Field& enemyField, const std::vector<ShipSize>& userShips, const std::vector<ShipSize>& enemyShips)**

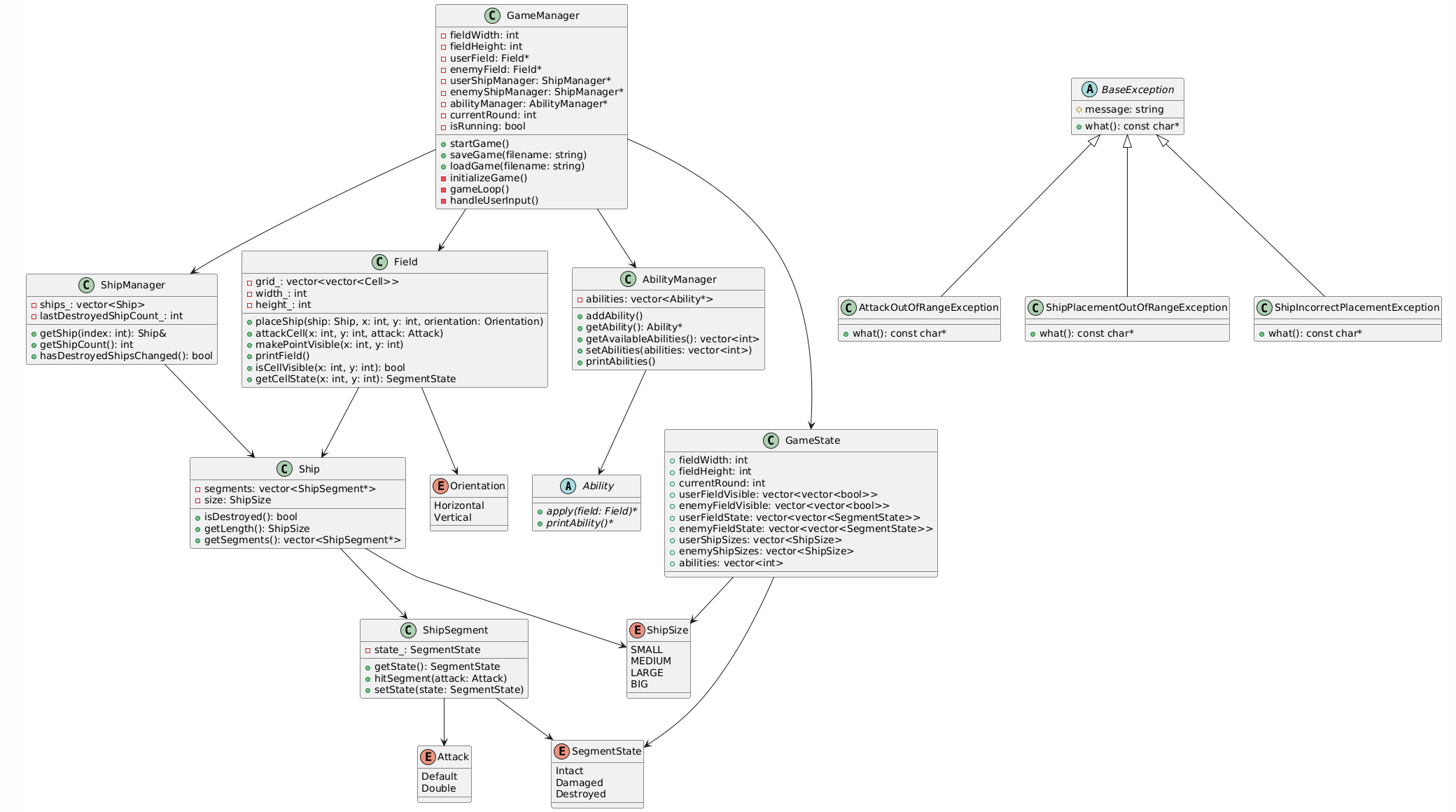
* **Описание:** Конструктор, который инициализирует объект GameState с заданными параметрами:
  + width и height — размеры поля.
  + round — текущий раунд игры.
  + userField и enemyField — поля игрока и противника.
  + userShips и enemyShips — списки размеров кораблей игрока и противника.

**friend std::ostream& operator<<(std::ostream& os, const GameState& state)**

* **Описание:** Перегруженный оператор вывода, который сериализует объект GameState в поток вывода.

**friend std::istream& operator>>(std::istream& is, GameState& state)**

* **Описание:** Перегруженный оператор ввода, который десериализует объект GameState из потока ввода.

Фигура 1: UML-диаграмма

**Вывод:**

В ходе выполнения лабораторной работы был реализован класс игры, который управляет игровым циклом, включая чередование ходов игрока и компьютерного врага, а также обработку побед и поражений. Были разработаны механизмы сохранения и загрузки игры, что позволяет игроку приостанавливать и возобновлять игру в любой момент.