МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе № 3

по дисциплине «Объектно-ориентированное программирование»

Тема: «Связывание классов»

Студент гр. 3343 _____ Жучков О.Д.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы

Изучить связывание классов, создать класс игры и реализовать сохранение/загрузку

Задание

- Создать класс игры, который реализует следующий игровой цикл:

i. Начало игры

ii. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

iii. В случае проигрыша пользователь начинает новую игру

iv. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

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.

- 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.

Примечание:

- Класс игры может знать о игровых сущностях, но не наоборот
- Игровые сущности не должны сами порождать объекты состояния
- Для управления самое игрой можно использовать обертки над командами
- При работе с файлом используйте идиому RAII.

Выполнение работы

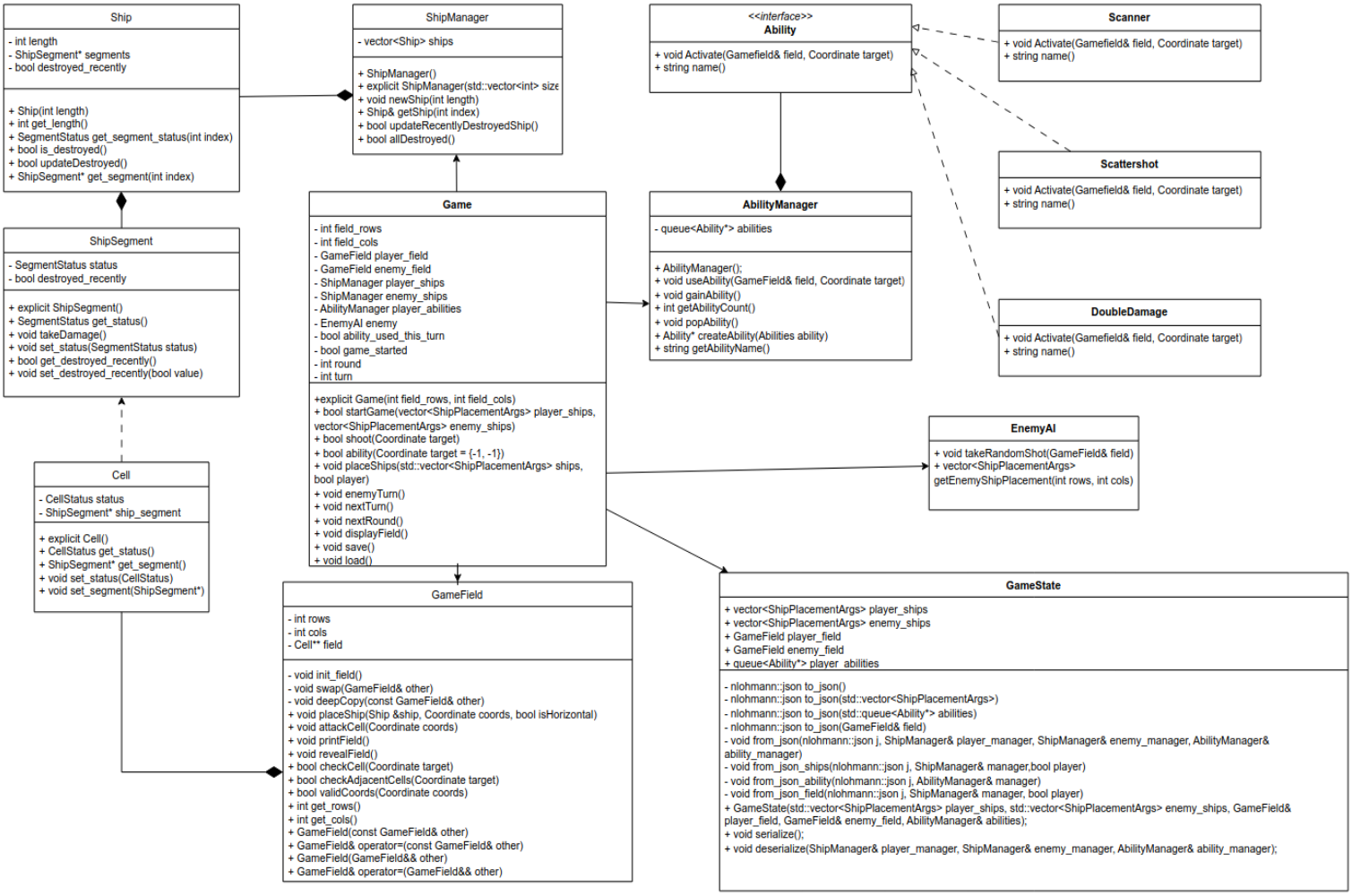

Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: Game, GameState.

Game — класс, реализующий основной игровой цикл, взаимодействие игрока с ним происходит через функции shoot, ability, save, load.

- int field_rows — ряды поля
- int field_cols — столбцы поля
- GameField player_field — поле игрока
- GameField enemy_field — поле врага
- ShipManager player_ships — корабли игрока
- ShipManager enemy_ships — корабли врага
- AbilityManager player_abilities — менеджер способностей
- EnemyAI enemy - «компьютерный соперник» для игрока

Методы:

- Game(int field_rows, int field_cols) - конструктор
- bool startGame(vector<ShipPlacementArgs> player_ships, vector<ShipPlacementArgs> enemy_ships) — Инициализация игрового процесса
- bool shoot(Coordinate target) — выстрел игрока по вражескому полю, после него либо начинается ход врага, либо начинается новый раунд
- bool ability(Coordinate target) — использование способности, возможно один раз за ход
- void placeShips(std::vector<ShipPlacementArgs> ships, bool player) — Расстановка кораблей по аргументам, относится к процессу инициализации
- void enemyTurn() - ход врага, выстрел по случайной клетке игрока
- void nextTurn() - начало следующего хода (вызвано в конце enemyTurn())
- void nextRound() - начало нового раунда
- void displayField() - вывод поля на экран
- void save() - сохранение
- void load() - загрузка

Класс GameState реализует сохранение и загрузку всех необходимых данных игры. Происходит сериализация и десериализация данных с помощью формата .json.

- GameState(std::vector<ShipPlacementArgs> player_ships, std::vector<ShipPlacementArgs> enemy_ships, GameField& player_field, GameField& enemy_field, AbilityManager& abilities) - конструктор
- void serialize() - сериализация и запись данных в json файл.
- void deserialize(ShipManager& player_manager, ShipManager& enemy_manager, AbilityManager& ability_manager) — десереализация данных и их загрузка в структуры игры

- nlohmann::json to_json() (и последующие одноименные функции) — преобразование данных объектов в json формат.
- void from_json(nlohmann::json j, ShipManager& player_manager, ShipManager& enemy_manager, AbilityManager& ability_manager) — выгрузка данных из json в объекты.

Вывод

Во время выполнения лабораторной работы созданы и связаны классы процесса игры и сохранения/загрузки.