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

Студентка гр. 3388 _____ Снигирев А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Объединить существующие классы в основной игровой цикл. Реализовать возможность сохранения и загрузки игры.

Задание

a) Создать класс игры, который реализует следующий игровой цикл:

1) Начало игры
2) Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
3) В случае проигрыша пользователь начинает новую игру
4) В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.
5) 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.

b) 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:

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

Выполнение работы

а) Игровой цикл

Файл `Game.h`. Содержит объявление класса `Game`. В полях класса лежат два объекта `ShipManager`, два объекта `GameField`, `AbilityManager` и переменные, отражающие текущее состояние игры.

Методы:

1. `void initialize_round();` - создает и случайным образом инициализирует корабли компьютера и размещает их на поле.
2. `void execute_player_turn();` - проведение хода игрока, во время хода игрок может сохранить/загрузить игру, использовать способность и провести обычную атаку.
3. `void execute_enemy_turn();` - проведение хода компьютера, атакует случайную точку поля игрока.
4. `bool check_victory() const;` - сравнивает количество уцелевших кораблей компьютера с 0.
5. `bool check_defeat() const;` - аналогично для игрока.
6. `void start_new_game();` - создание и инициализация поля и менеджера игрока, запуск игрового цикла.
7. `void start_round();` - игровой цикл, где игрок и компьютер ходят по очереди до тех пор, пока у кого-нибудь не кончатся корабли. Если это будет игрок, игра перезапустится, если компьютер — игра перейдет на новый раунд.
8. `void save_game(const std::string& filename);` - сохраняет текущее состояние игровых сущностей с помощью метода класса `GameState`.
9. `void load_game(const std::string& filename);` - загружает из файла данные и ини присваивает игровым сущностям данные сохранения.

10. `void UseCommand();` - вспомогательный метод, отвечающий за обработку команды от игрока, вызывается из метода-хода игрока.
11. `void UseAbility();` - вспомогательный метод, отвечающий за применение способности игроком, вызывается из метода-хода игрока

б) Класс состояний. Создается в момент сохранения и загрузки, при сохранении инициализируется текущими игровыми сущностями, при загрузке — сущностями, которые сохранены в файл.

GameState:

`int roundNumber;` - номер раунда.

`GameField pField;` - поле игрока

`GameField eField;` - поле компьютера

`std::vector <char> pAbilities;` - символьное представление очереди способностей.

`ShipManager pShips;` - корабли игрока

`ShipManager eShips;` - корабли противника

Методы:

1. `GameState(const int round, const GameField& playerField, const GameField& enemyField, std::vector<char> playerAbilities);`
2. `friend std::ostream& operator<<(std::ostream& out, const GameState& state);` - переопределение оператора вывода. При использовании должен записывать состояние игры в файл.
3. `friend std::istream& operator>>(std::istream& in, GameState& state);` - переопределение оператора ввода. При использовании должен считывать состояние игры из файла.
4. `void save(std::ostream& out, const ShipManager& playerShips, const ShipManager& enemyShips);` - функция, производящая сохранения игры.

5. `void load(int round, GameField& playerField, GameField& enemyField,ShipManager& playerShips, ShipManager& enemyShips, AbilityManager& playerAbilities, std::istream& in);` - функция, производящая загрузку игры.

Класс ввода. `Input`. Вспомогательный класс, отвечающий за начальный ввод игроком кораблей и их размещения.

UML-диаграмму классов смотреть в **ПРИЛОЖЕНИИ**

Вывод

В ходе лабораторной работы был добавлен функционал использования особых способностей. Также были написаны классы-исключения для наиболее распространенных предполагаемых исключений и сделаны корректировки имеющейся логики.

ПРИЛОЖЕНИЕ

UML-диаграмма

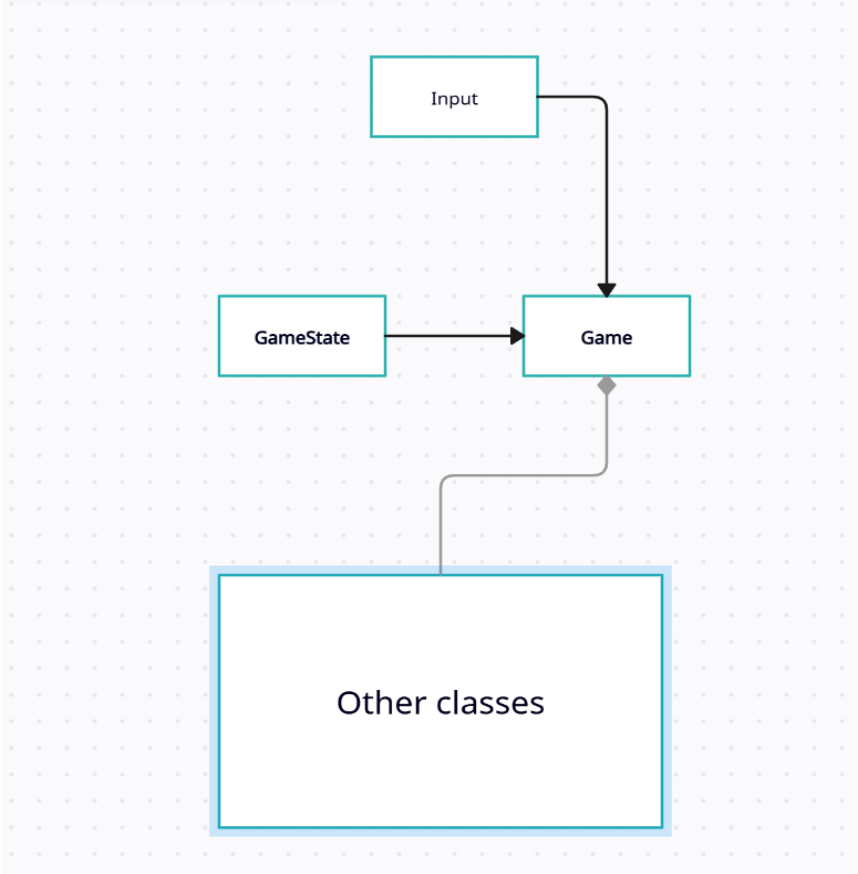

Рисунок 1. UML-диаграмма