**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. 3384 |  | Пьянков М.Ф. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

**2024**

## Цель работы

Изучить работу с сохранением объектов на c++ в файл. Применить обёртку RAII.

## 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

Архитектура новых модулей представлена на рис. 1.

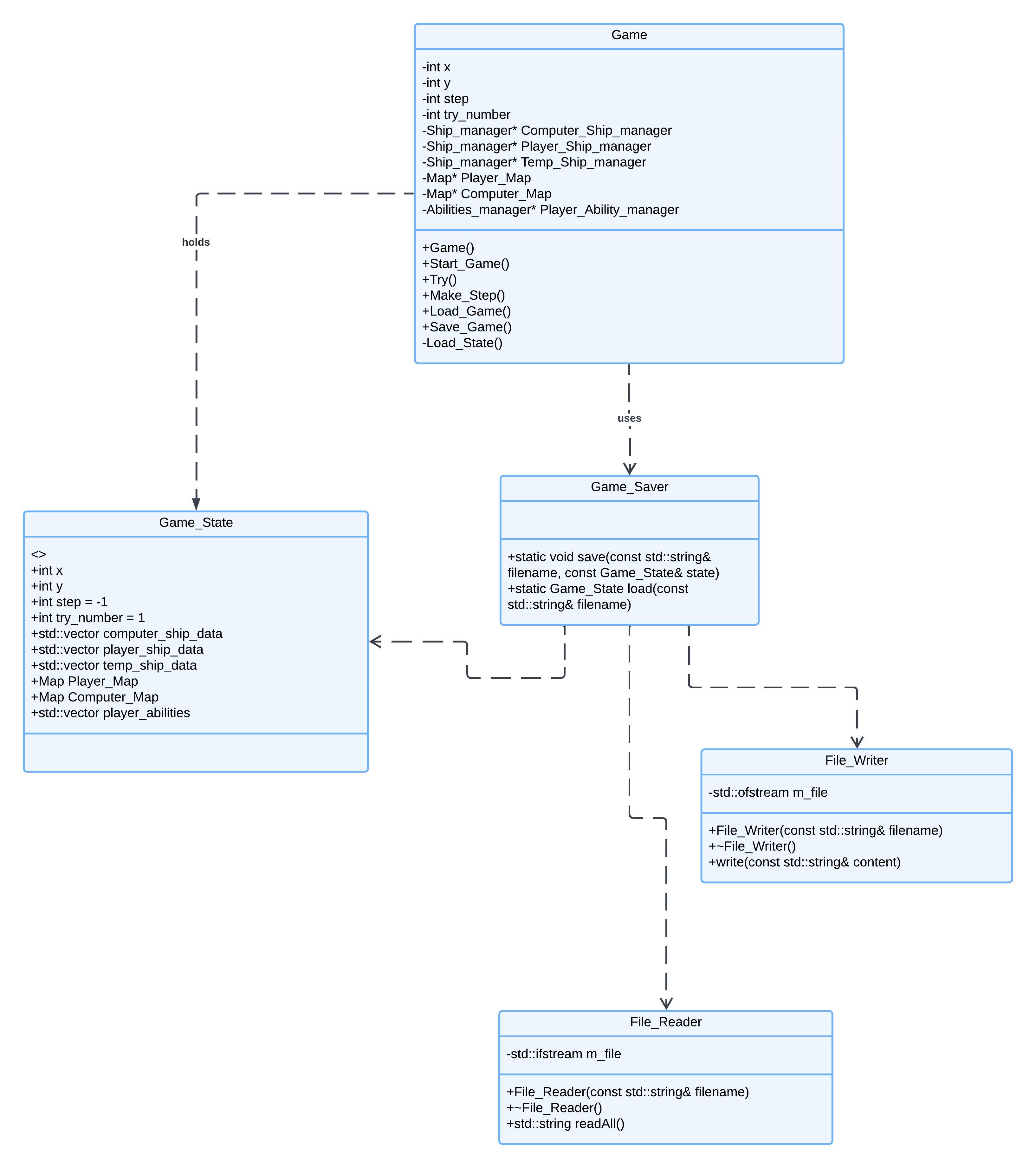


Рисунок 1 — Архитектура новых модулей программы (UML-диаграмма классов)

класс Game основной класс

Методы: void Start\_Game(), метод начала новой попытки: void Try(), метод загрузки и сохранения игры: void Load\_Game(), void Save\_Game(), вспомогательный метод загрузки структуры состояния Game\_state: void Load\_State.

Поля: менеждеры кораблей всех участников и буферный менеджер: Ship\_manager\* Computer\_Ship\_manager, Ship\_manager\* Player\_Ship\_manager, Ship\_manager\* Temp\_Ship\_manager; карты игрока и компьютера: Map\* Player\_Map, Map\* Computer\_Map; менеджер способностей игрока: Abilities\_manager\* Player\_Ability\_manager; Структура состояния игры и её загрузчик: Game\_State state, Game\_Saver game\_saver; размеры карты: int x, int y; текущий ход и номер раунда: bool step, int try\_number.

Структура Game\_State – содержит поля, хранящие необходимые для загрузки и восстановления данные.

Поля: размер карты:int x, int y; текущий ход: bool step; номер текущего раунда: int try\_number; векторы кораблей игрока, компьютера и буфера: std::vector<Ship>computer\_ship\_data, std::vector<Ship>player\_ship\_data, std::vector<Ship>temp\_ship\_data; карты игрока и компьютера: Map Player\_Map Map(0, 0), Map Computer\_Map = Map(0, 0); список способностей игрока: std::vector<enum ABILITY> player\_abilities.

Класс Game\_Saver – класс, реализующий сохранение и загрузку Game\_State.

Метод: save и load – сохранение и загрузка соотв.

Классы File\_Reader и File\_Writer – обёртка над базовыми методами ввода-вывода в файл (RAII) реализуют соответствующие методы, с учётом закрытия поток ввода и вывод в случае ошибки.

Также для всех объектов написаны функции их сериализации и десериализации. Сохранение и загрузка производятся из файла формата JSON.

## Выводы

Успешно реализован класс Game, описана структура состояния игры Game\_State, выполнена обёртка RAII для работы с файлами, созданы сопуствующие классы.