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

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

b) Начало игры

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

d) В случае проигрыша пользователь начинает новую игру

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

f) 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.

g) 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.

h) Примечание:

i) Класс игры может знать о игровых сущностях, но не наоборот

j) Игровые сущности не должны сами порождать объекты состояния

k) Для управления самое игрой можно использовать обертки над командами

l) При работе с файлом используйте идиому RAII.

Выполнение работы

Архитектура новых модулей представлена на рис. 1.

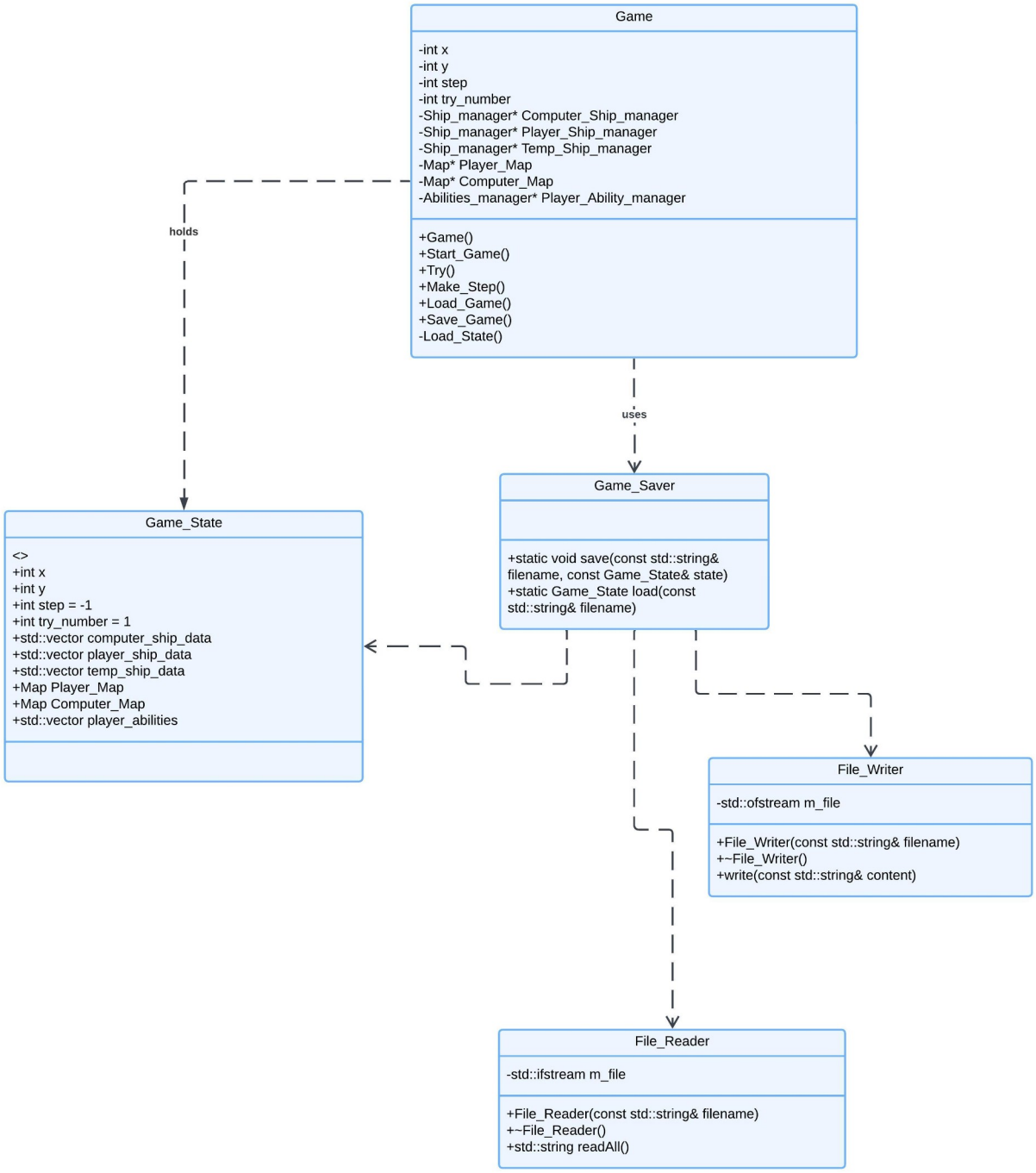

Рисунок 1 — Архитектура новых модулей программы (UML-диаграмма классов)

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: менеджеры 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.