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. 3383 _____ Боривец С. Ю.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы

Создать класс игры, который реализует игровой цикл, а также класс состояния игры. Реализовать сохранение и загрузку игры. Продумать архитектуру проекта, учитывая результаты предыдущей работы.

Задание

Лабораторная работа №3 - Связывание классов

Создать класс игры, который реализует следующий игровой цикл:

Начало игры

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

В случае проигрыша пользователь начинает новую игру

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

Класс игры(Game) – реализует игровой цикл:

У данного класса есть поля:

Game_state – отвечает сохранение и загрузку состояния игры;

Player_gamefield – хранит поле игрока;

Computer_gamefield – хранит поле компьютера;

Player_ship_manager – хранит менеджер кораблей игрока;

Computer_ship_manager – хранит менеджер кораблей компьютера;

Player_ability_manager – хранит менеджер способностей игрока;

Метод new_game() отвечает за начало новой игры. Создаются новые поля и менеджеры.

Метод round() отвечает за чередование раундов. Сначала у пользователя спрашивают, хочет ли он сохранить игру(метод game_save()) или загрузить игру(метод game_load()). После идет ход игрока и компьютера. В методе player_move() реализован ход игрока, а в методе computer_move() прописан алгоритм действий компьютера.

Класс игры связывает все классы. Делает так, чтобы они взаимодействовали друг с другом.

Класс состояния игры(Game_state) – реализует класс состояния игры. С помощью его можно сохранять и загружать игру.

Для избегания дублирования кода объявляются макросы с названиями объектов поля и менеджеры игрока и компьютера. Также для этого создается enum Datapack. С его помощью мы можем просто доставать нужный нам пакет данных об игровом объекте из класса Game_state.

В самом классе Game_state объявлены приватные поля строк с игровыми объектами. Вся информация будет сохраняться в них. Также есть поле, которое хранит путь на файл сохранения.

В публичном пространстве доступа находятся следующие методы:

Конструктор, который принимает на вход ссылку на файл, а затем сразу инициализирует полученный ресурс(RAII).

Метод установки пути set_path_to_save_file() – принимает на вход пусть, по умолчанию, как и в конструкторе, стоит путь "../saves/slot1.txt".

Метод load_game_state() – загружает из txt файла, в котором хранится сохранение, игру.

Метод save_game_state() - принимает на вход игровые объекты и сохраняет их в txt файл.

Метод get_datapack() – принимает на вход enum-значение определяющим нужный пакет данных об игровом объекте.

Метод clear() – полностью отчищает данные об игровых объектах в game_state.

Сохранение реализовано таким образом: игровые объекты записываются в следующем порядке: менеджер кораблей игрока, поле игрока, менеджер способностей игрока, менеджер кораблей компьютера и поле компьютера. Каждому сохраняемому игровому классу был написан метод(ship_manager_to_string, gamefield_to_string, ability_manager_to_string), который сохраняет данные об объекте путем перевода объекта в строку. Таким образом, сначала в файл сохранения записывается заголовок объекта, а затем его строковое сохранение, так происходит с каждым игровым объектом в указанном ранее порядке.

Загрузка реализована следующим образом: с помощью метода load_game_state класс состояния игры считывает данные о каждом игровом объекте в порядке: менеджер кораблей игрока, поле игрока, менеджер способностей игрока, менеджер кораблей компьютера и поле компьютера. При загрузке класс состояния игры обращает внимание на заголовок, сохраняет загруженные данные(строки с сохраненными данными) в полях, предназначенных для них. Каждому загружаемому игровому классу был прописан метод(load), который обрабатывает данные из полученной им строки.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Создано поле игрока, расставлены корабли на нем. Создано поле компьютера, расставлены корабли на нем.	Игровой цикл работает. При победе и поражении игра перезапускается, можно сохранить, а также загрузить игру.	OK

Выводы

Созданы класс игры, который реализует игровой цикл, а также класс состояния игры. Реализовано сохранение и загрузка игры. Архитектура проекта продумана с учётом результатов предыдущей работы.

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА КЛАССОВ