МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Объектно-ориентированное программирование»
Тема: Сериализация состояния программы

Студентка гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Реализация сохранения и загрузки программы.

Задание.

Основные требования:

- Возможность записать состояние программы в файл
- Возможность считать состояние программы из файла
- Загрузка и сохранение должно выполняться в любой момент программы
- Взаимодействие с файлами должны быть по идиоме RAII
- Сохранение и загрузка реализованы при помощи паттерна “Снимок”
- Реализован контроль корректности файла с сохраненными данными

Реализация сохранения и загрузки состояния программы

Реализация сохранения и загрузки состояния программы в любой момент времени реализована следующим образом:

- Facade имеет методы сохранения и загрузки программы(saveGame, loadGame), которые могут быть вызваны UI в любой момент времени.
- Загрузка и сохранения реализованы по принципу паттерна «Снимок». Класс Game возвращает снимок(GameMemento) при считывании состояния программы в методе readMemento(string), а так же создает снимок при записи состояния в методе restoreMemento(Memento *). Для передачи информации использованы дополнительные классы.
- Для сохранения и загрузки в файл по идиоме RAII созданы классы WriteToFile, ReadFromFile, которое проверяют корректность файла с сохраненными данными.

Классы для сохранения и считывания состояния программы

Для передачи и сохранения различных состояний игры введены дополнительные классы: GameParam, BaseParam, UnitParam, NeutralParam. Поля данных классов подбронее описаны в табл. 1-4.

Таблица 1 - Поля класса GameParam

Поле	**Назначение**
`unsigned width;`	Значение ширины поля
`unsigned height;`	Значение высоты поля
`vector<BaseParam*> bases;`	Массив информации о базах на поле
`vector<NeutralParam*> neutrals;`	Массив информации о нейтральных объектах на поле
`vector<LandscapeType> landscape;`	Массив ландшафта поля

Таблица 2 - Поля класса BaseParam

Поле	**Назначение**
`int baseNumb;`	Номер базы
`int unitCount;`	Текущие значение юнитов в базе
`const int maxCount;`	Значение максимального кол-ва юнитов
`int health;`	Значение здоровья базы
`const int x, y;`	Координаты базы на поле
`vector<UnitParam*> units;`	Массив юнитов, принадлежащих базе

Таблица 3 - Поля класса UnitParam

Поле	**Назначение**
`string name;`	Название юнита
`int baseNumber;`	База, к которой принадлежит юнит
`Attributes* attributes;`	Значение аттрибутов юнита

Таблица 4 - Поля класса NeutralParam

Поле	**Назначение**
`NeutralType type;`	Тип нейтрального объекта
`unsigned x;`	Значение координаты X на поле
`unsigned y;`	Значение координаты Y на поле

Классы взаимодействия с файлами

Для сохранения и загрузки в файл по идиоме RAII созданы классы WriteToFile, ReadFromFile, которое проверяют корректность файла с сохраненными данными.

- В классе WriteToFile для чтения из файла используется класс ifstream, в ReadFromFile для записи – std::ofstream.
- В обоих классах файл открывается в конструкторе (при ошибке бросается соответствующее исключение), а закрывается в деструкторе.

Методы классов представлены в табл.5.

Таблица 5 — Методы взаимодействия с файлами

Метод	**Назначение**
`void write(GameParam* params);`	Метод записи в файл, принимает на вход класс параметров игры, записывает в определнном формате все параметры.
`GameParam* read();`	Метод считывания из файла, возращает класс параметров игры со считанными данными

Контроль корректности файла реализован следующим образом:

- Разделы в файле должны иметь заголовки: например, перед разделом информации о поле должен быть заголовок «Field», перед разделом с нейтральными объектами – «Neutrals». Проверка осуществляется в методе read() класса MementoReader.
- В том же методе имеется контроль корректности численных значений параметров. В случае ошибки (недопустимые символы, числа) бросается исключение, загрузка игры в конечном счете не осуществляется.

Снимок

Класс GameMemento (файлы gamememento.h/cpp) реализован по принципу *паттерна «Снимок»*. Основные методы и конструкторы класса приведены в табл. 6.

Таблица 6 — Основные методы класса GameMemento

Метод	**Назначение**
`GameMemento(string name, GameParam* params);`	Конструктор для сохранения состояния программы. С помощью класса WriteToFile в файл с именем name записывается состояние, определенное params
`GameMemento(string name);`	Конструктор для последующей загрузки из файла. В теле конструктора загрузка не выполняется
`GameParam* loadFromFile();`	Метод загрузки состояния из файла. Имя файла определяется в конструкторе, информация считывается с помощью класса ReaderFromFile

Паттерн «Снимок» также предполагает наличие «Создателя», которым является класс Game (файлы game.h/cpp), в котором были добавлены методы для сохранения и загрузки игры. Особенности методов описаны в табл. 7.

Таблица 7 — Методы загрузки и сохранения класса Game

Метод	**Назначение**
`void restoreMemento(Memento* memento);`	Метод создает объект снимка и заполняет его данными из поля (Field) и баз (Base). Практически все параметры уже имеют необходимый тип, преобразования требуются только для перечислений

Продолжение таблицы 7

`GameMemento* readMemento(string name);`	Метод по объекту снимка последовательно восстанавливает состояние игры в следующем порядке: • Удаляются старые данные об игре • Создается новое поле и посредник для хранения и взаимодействия юнитов • Поле заполняется ландшафтом • На поле помещаются нейтральные объекты • На поле последовательно добавляются базы и их юниты (то есть база, ее юниты, потом следующая база, ее юниты, и так далее). После создания, юнитам присваиваются сохраненные характеристики

Выводы

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для сохранения и загрузки состояния программы. Были использованы паттерны проектирования, а также принципы объектно-ориентированного программирования.