**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. 3342 |  | Иванов С.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Написать класс игры и класс состояния игры. Также реализовать сохранения состояния игры путём сериализации классов.

## **Задание**

1. 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.
2. 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.
3. 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.

## **Выполнение работы**

Класс Serializer:

Абстрактный класс сериализатора, что наследуется от Visitor.

Класс содержит следующие публичные методы:

* void save(Owner \*\_ow); - метод, отвечающий за сериализацию объектов;
* const nlohmann::json& get() const noexcept; - метод, который выдаёт сериализованный объект;
* ~ Serializer() override = default; - переопределённый деструктор;

Все наследники этого класса сериализуют объекты определённых классов, а именно:

* SkillManagerSerializer - для SkillManager;
* UnitSerializer - для Unit;
* FieldSizeSerializer - для Field::Size;
* ShipSerializer - для Ship;
* GameStateSerializer - для GameState;
* ShipManagerSerializer - для ShipManager;
* FieldSerializer - для Field;

Класс Loader:

Абстрактный класс загрузчика.

Класс содержит следующие методы:

* virtual void load(const nlohmann::json &\_json) = 0; - метод, определяющий логику загрузки;
* std::shared\_ptr<Owner> get() const noexcept; - метод, который выдаёт загруженный объект;
* ~ Loader() override = default; - переопределённый деструктор;

Все наследники этого класса загружают объекты определённых классов, а именно:

* SkillManagerLoader - для SkillManager;
* UnitLoader - для Unit;
* FieldSizeLoader - для Field::Size;
* ShipLoader - для Ship;
* GameStateLoader - для GameState;
* ShipManagerLoader - для ShipManager;
* FieldLoader - для Field;

Класс File:

Класс для указания и хранения имени файла.

Класс содержит следующие публичные методы:

* const std::string& name(const std::string &name) noexcept; - сеттер имени файла;
* const std::string& name() const noexcept; - геттер имени файла;

Класс GameState:

Класс состояния игры.

Класс содержит следующие публичные методы:

* explicit GameState(const Field &\_player\_field, const Field &\_enemy\_field, const ShipManager &\_player\_ship\_manager, const ShipManager &\_enemy\_ship\_manager, const SkillManager &\_player\_skill\_manager);
* explicit GameState(const Field &\_player\_field, const Field &\_enemy\_field);
* GameState();
* GameState(const GameState &other);
* GameState(GameState &&other) noexcept;
* ~GameState() = default;
* GameState& operator=(const GameState& other);
* GameState& operator=(GameState&& other) noexcept;
* Field& get\_player\_field() noexcept; - геттер поля игрока
* Field& get\_enemy\_field() noexcept; - геттер поля противника
* ShipManager& get\_player\_ship\_manager() noexcept; - геттер менеджера кораблей игрока
* ShipManager& get\_enemy\_ship\_manager() noexcept; - геттер менеджера кораблей врага
* SkillManager& get\_player\_skill\_manager() noexcept; - геттер менеджера скилов игрока
* void set\_player\_field(const Field &\_player\_field) noexcept; - сеттер поля игрока
* void set\_enemy\_field(const Field &\_enemy\_field) noexcept; - сеттер поля противника
* void set\_player\_ship\_manager(const ShipManager &\_player\_ship\_manager) noexcept; - сеттер менеджера кораблей игрока
* void set\_enemy\_ship\_manager(const ShipManager &\_enemy\_ship\_manager) noexcept; - сеттер менеджера кораблей врага
* void set\_player\_skill\_manager(const SkillManager &\_player\_skill\_manager) noexcept; - сеттер менеджера скилов игрока

Класс Game:

Класс игры.

Класс содержит следующие публичные методы:

* Game();
* void user\_attack(const Unit & \_unit); - атака пользователя
* SkillResultStatus user\_skill(const Unit &\_unit = Unit()); - использование скила, если он есть
* void load\_game(); - загрузка игры
* void save\_game(); - сохранение игры
* void new\_game(); - новая игра
* void bot\_attack(); - атака бота
* void setup\_bot(); - пересборка поля и менеджера кораблей бота
* GAME\_OVER\_FLAG is\_game\_over() noexcept; - проверка на победителя
* GameState& state() noexcept; - геттер состояния игры
* void set\_state(GameState & \_state) noexcept; - сеттер состояния игры
* ~Game() = default;

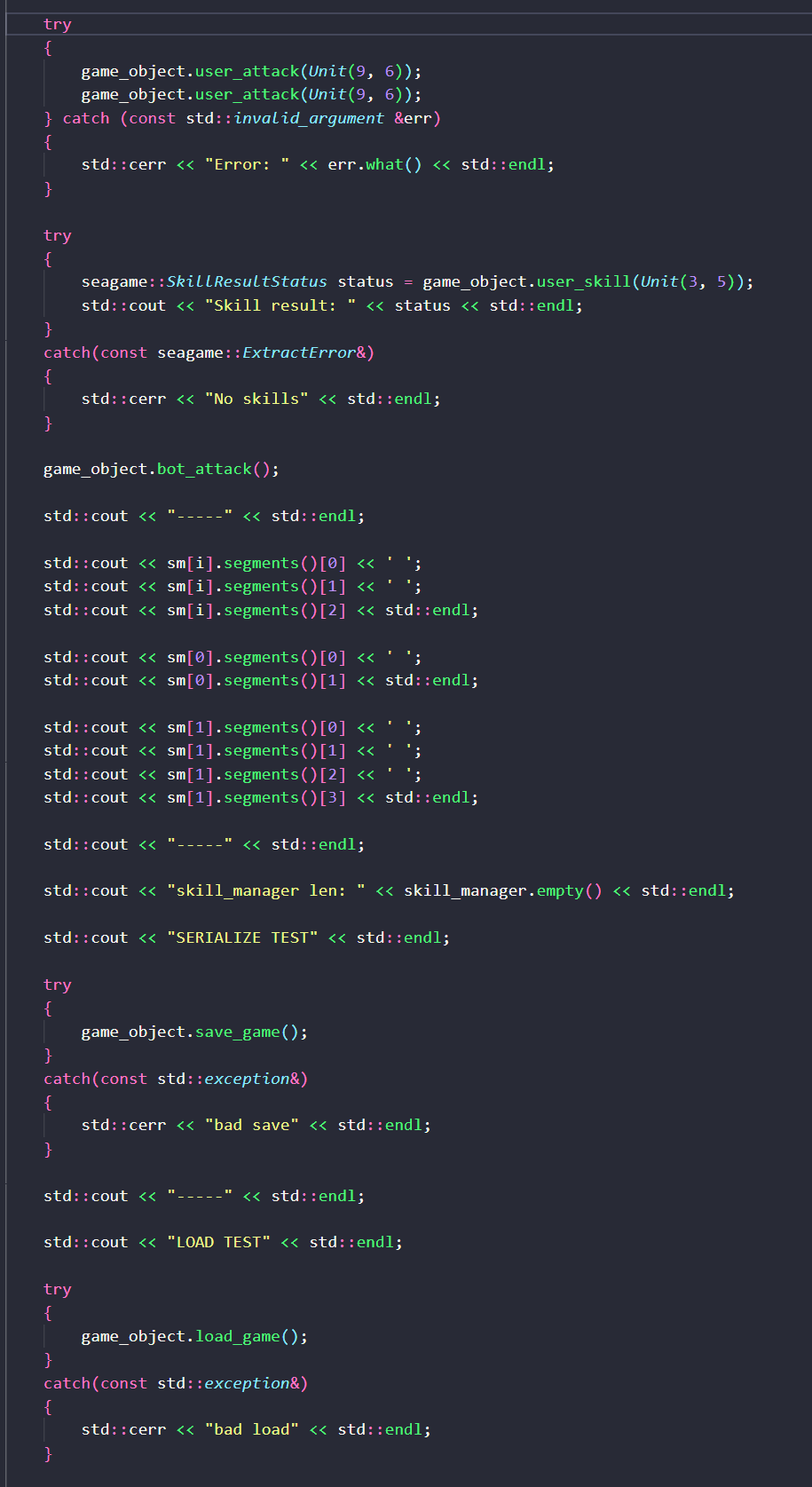
Архитектурные решения:

Были реализованы класс состояния игры, который отвечает за ресурсы игры и её состояния, также класс игры, который отвечает за процесс игры и реализует функционал, нужный для игрового цикла.

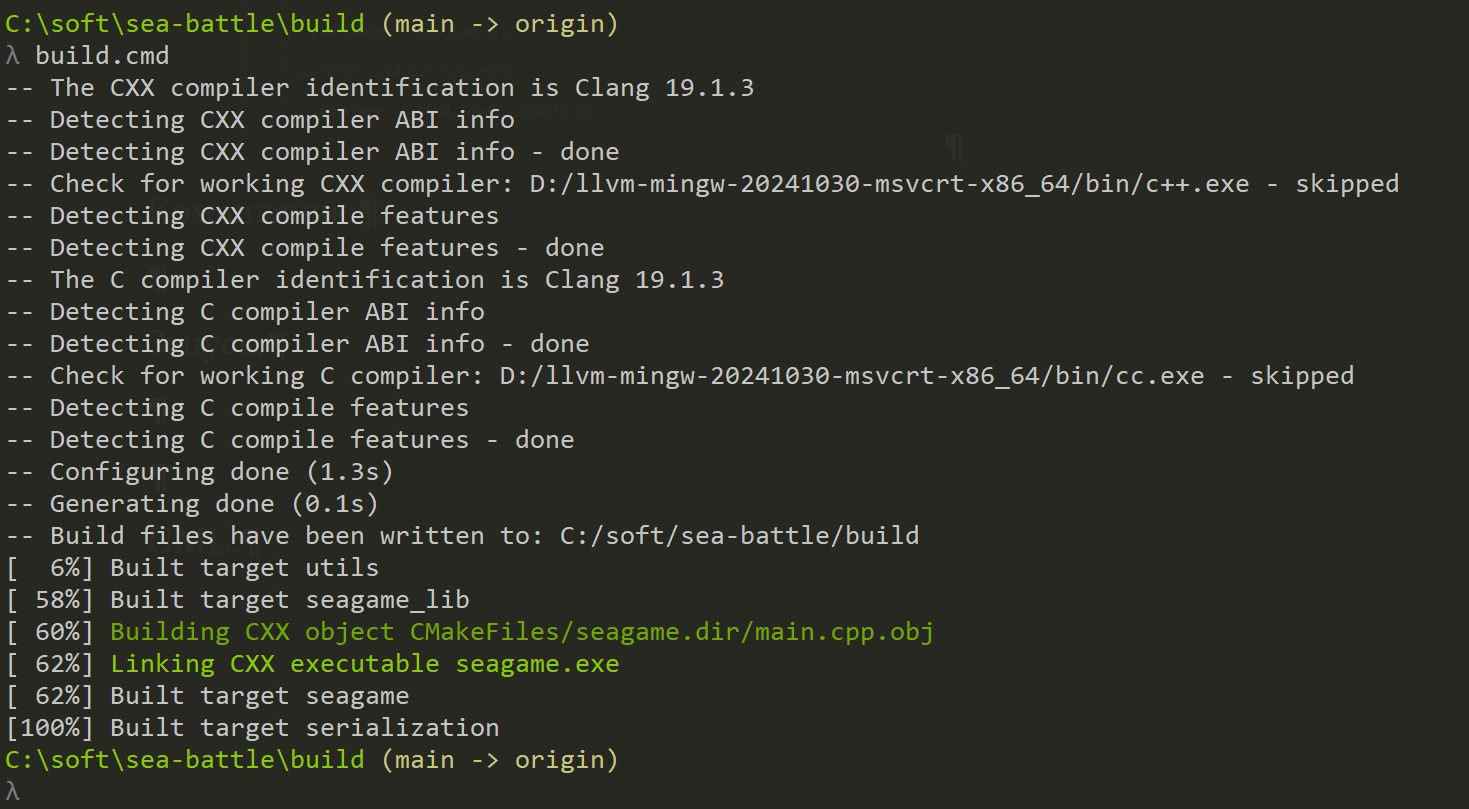
Также были записаны классы-серализаторы и классы-лоадеры, которые отвечают за сохранение-загрузку данных игры. Было решено поместить их в отдельную динамическую библиотеку и подгружать их по необходимости, так как они не реализуют основной функционал игры.

Проверка работоспособности написанного кода:

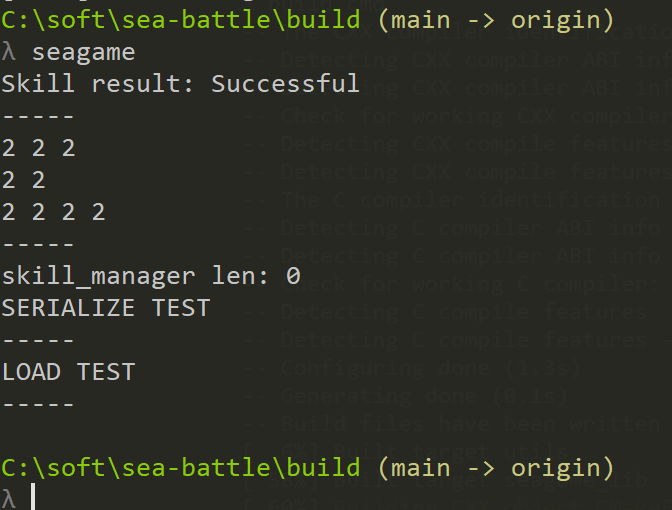




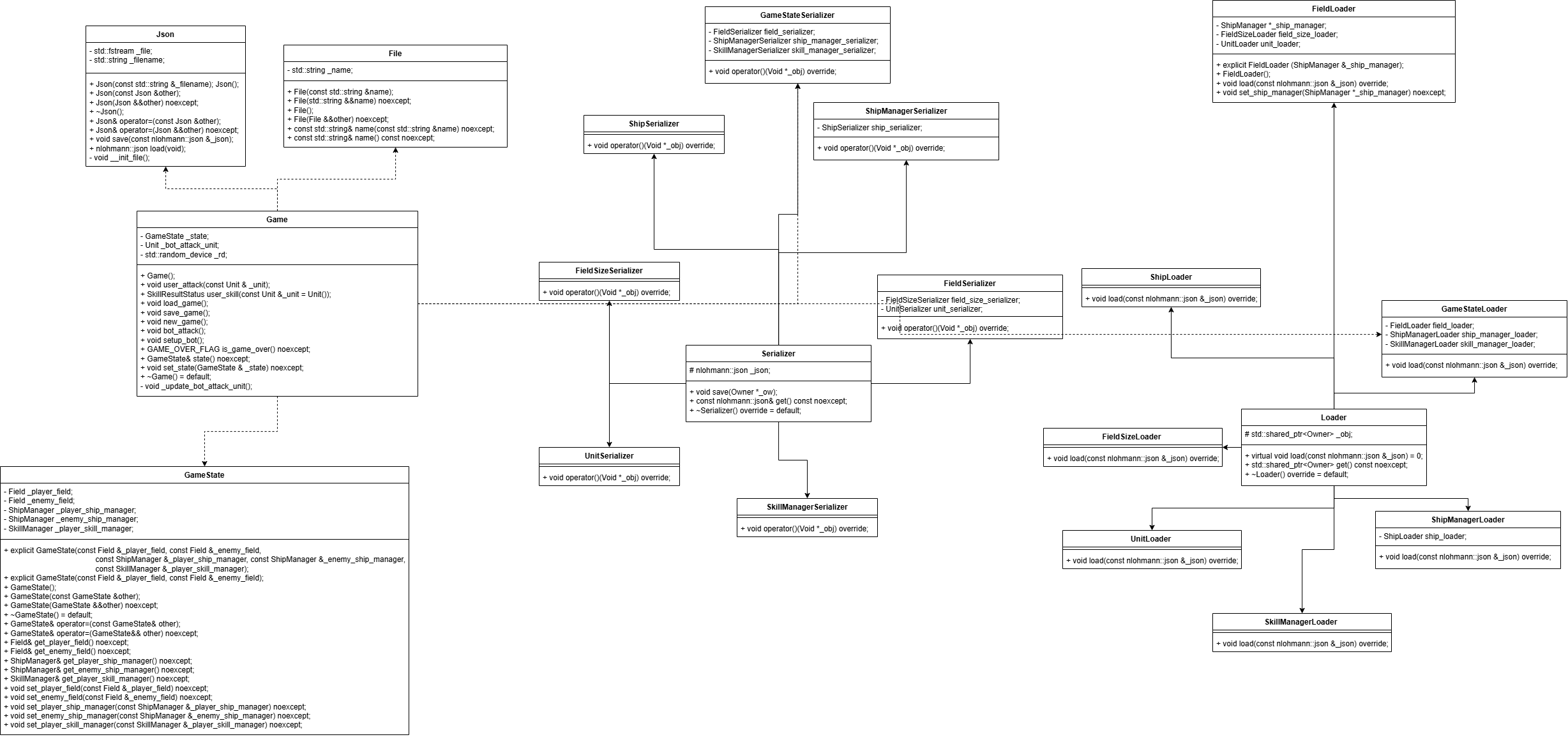
Компиляция:



Запуск:



UML:



## **Выводы**

В ходе выполнения работы были написаны класс игры и класс состояния игры. Также реализовано сохранения состояния игры путём сериализации классов.