**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3383 |  | Земерова С.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург 2025

# Цель работы

Создать класс игры, который реализует игровой цикл, а также класс состояния игры с переопределенными операторами ввода и вывода в поток. Также необходимо сделать возможность сохранения и загрузки игры.

# 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.

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.

1. 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.

# Описание архитектурных решений и классов

В ходе данной лабораторной работы был создан класс игры (Game), класс состояния игры (GameState), класс обёртки для работы с файлом (FileWrapper), обстрела (Damage), класс сохранения поля (SaverField), класс сохранения очереди способностей (SaverAbility), класс сохранения менеджера кораблей (SaverManager).

**Класс Game** – представляет собой основную игровую логику морского боя. Класс управляет игровым процессом, взаимодействием пользователя и системы. Позволяет загружать и сохранять игру, организовывать раунды и обрабатывать ходы игроков и противника.

*Game*() – конструктор класса. Приветствует пользователя и предлагает загрузить сохранённую игру или начать новую. В случае отказа от загрузки, запускает новый игровой процесс.

Методы:

*void startGame()* – инициализирует новую игру. Настраивает игровое поле, размещает корабли пользователя и противника, а затем запускает первый раунд. *void setField()* – устанавливает размеры игрового поля. Обрабатывает ввод

пользователя и проверяет данные на корректность.

*void setNumShips()* – задаёт количество кораблей и их размеры. Выполняет проверку на соответствие размеров поля и обрабатывает исключения.

*void setUserShips()* – размещает корабли пользователя на игровом поле. Обрабатывает ввод координат и ориентации кораблей. Реагирует на ошибки размещения, выдавая соответствующие сообщения.

*void setEnemyShips()* – автоматически размещает корабли противника. Использует генерацию случайных координат и ориентации, учитывая возможность ошибок и ограничений.

*void startRound() –* организует игровой раунд, в котором игрок и противник совершают ходы. Проверяет состояние игры на завершение раунда или начало нового.

*void userMove()* – обрабатывает ход пользователя. Предоставляет возможность сохранить и загрузить игру, использовать способности и выполнить атаку.

*void enemyMove()* – обрабатывает ход противника, включая автоматическую генерацию координат атаки.

*void saveGame() –* сохраняет текущее состояние игры в файл. Использует класс *GameState* для сериализации данных.

*void loadGame() –* загружает сохранённое состояние игры из файла.

Инициирует восстановление игровых объектов.

*char getRandomOrientation() –* возвращает случайную ориентацию ('h' или 'v') для размещения корабля.

*std::vector<int> getRandomCoordinates()* – генерирует случайные координаты для размещения или атаки.

**Класс GameState** отвечает за сохранение и загрузку состояния игры. Этот класс обеспечивает сериализацию и десериализацию игровых данных, таких как поля, менеджеры кораблей, способности и текущий раунд. Его основной задачей является управление взаимодействием с файлами для сохранения и восстановления игры.

Методы:

*GameState(Field& uf, Field& ef, Manager& um, Manager& em, AbilityManager& am, int cr)* – конструктор, инициализирующий состояние игры. Принимает текущее состояние игрового поля, менеджеров и способностей, а также номер текущего раунда.

*void saveToFile(const std::string &filename) const* – сохраняет состояние игры в файл. Использует сериализацию с помощью операторов ввода и вывода и обработку ошибок. При успешном сохранении выводит сообщение. В случае ошибки выводит соответствующее сообщение.

*void loadFromFile(const std::string &filename)* – загружает состояние игры из файла. Восстанавливает данные полей, менеджеров, способностей и текущего раунда из файла.

*std::istream &operator>>(std::istream &in, GameState &state)* – оператор перегрузки ввода. Десериализует данные из потока: текущий раунд, состояния полей, состояния менеджеров, очередь способностей.

*std::ostream &operator<<(std::ostream &out, const GameState &state)* – оператор перегрузки вывода. Сериализует данные в поток: текущий раунд, состояния полей, состояния менеджеров, очередь способностей.

**Класс FileWrapper** – отвечает за безопасное открытие, использование и закрытие файлов. Используется для управления файловыми потоками и предотвращения утечек ресурсов, таких как незакрытые файлы. Класс инкапсулирует работу с std::fstream и обеспечивает автоматическое закрытие файла при завершении работы объекта.

Методы:

*FileWrapper(const std::string& filename, std::ios::openmode mode)* – конструктор, который открывает файл с указанным именем *filename* и режимом *mode* (например, *std::ios::in* для чтения или *std::ios::out* для записи). Если файл не может быть открыт, генерируется исключение *std::ios\_base::failure* с описанием ошибки.

*~FileWrapper()* – деструктор. Гарантирует, что файл будет закрыт, если он был открыт, даже в случае возникновения ошибок.

*std::fstream& get()* – возвращает ссылку на объект *std::fstream*, обеспечивая доступ к открытому файловому потоку.

**Класс SaverField** предназначен для сохранения и загрузки состояния игрового поля Field в поток (например, файл) или из потока. Основная цель — сериализация и десериализация объектов типа Field.

Методы:

*static void save(std::ostream& out, const Field& field)* - сохраняет состояние поля Field в выходной поток out. Сохраняются размеры поля (width и height).

Итерируется по каждой ячейке поля: если в ячейке находится корабль, записываются: координаты ячейки (y, x), длина корабля, ориентация корабля (0 для горизонтальной, 1 для вертикальной), Состояние сегмента (*segState*) из тела корабля (*shipBody*), индекс сегмента (*indSeg*). В конце записывается маркер окончания поля: FIELD\_END .

*static void load(std::istream& in, Field& field) -* загружает состояние поля Field из входного потока in. Считываются размеры поля (*width* и *height*) и создаётся новое поле с этими параметрами. Считываются данные о каждом корабле. Создаётся объект Ship и добавляется в соответствующую ячейку поля. Считывание завершается, когда встречается маркер FIELD\_END .

Конструктор *SaverField::SaverField(const Field& field) -* сохраняет ссылку на объект Field, если требуется создать экземпляр SaverField для работы с конкретным полем.

**Класс SaverManager -** реализует сериализацию и десериализацию объектов класса Manager. Этот класс используется для сохранения состояния менеджера (оставшихся кораблей и их размеров) и восстановления его состояния из файла.

Методы:

*static void save(std::ostream& out, const Manager& manager)* - сохраняет состояние объекта Manager в поток out. Записывается маркер начала менеджера:

MANAGER\_START . Сохраняется количество оставшихся кораблей (*getRemainingNumShips*) и список размеров кораблей (*getSizeShips*).

*static void load(std::istream& in, Manager& manager)* - загружает состояние объекта Manager из потока in. Пропускаются строки до маркера

MANAGER\_START . Считывается количество оставшихся кораблей, список размеров кораблей. Создаётся объект Manager с полученными параметрами.

**Класс SaverAbility** обеспечивает сериализацию и десериализацию очереди способностей *std::queue<std::shared\_ptr<IAbility>>*. Используется для

сохранения текущих способностей игры и их восстановления из сохранённого файла.

Маппинг типов способностей:

*AbilityTypeToString* - cопоставляет перечисления AbilityType строковым значениям.

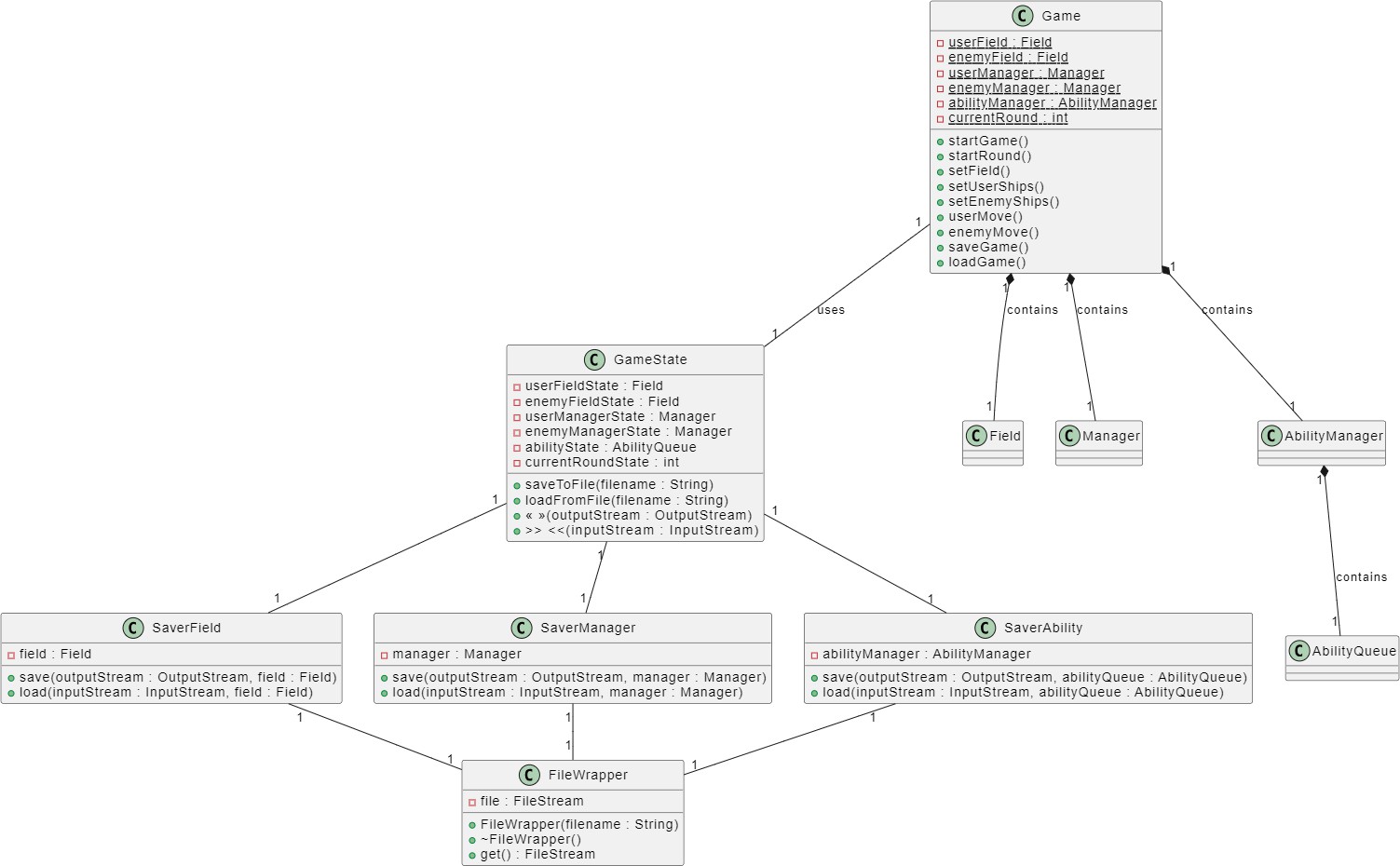
*StringToAbilityType -* обратное сопоставление для загрузки из строкового представления.

Методы:

*static void save(std::ostream& out, std::queue<std::shared\_ptr<IAbility>>& abilityQueue)* - cохраняет очередь способностей в поток out. Получает текущий размер очереди. Итерируется по всем элементам очереди: извлекает способность, сохраняет её тип в строковом формате (AbilityTypeToString), после чего возвращает способность обратно в очередь.

*static void load(std::istream& in, std::queue<std::shared\_ptr<IAbility>>& abilityQueue)* - загружает очередь способностей из потока in. Считывает строки, пока поток не закончится. Для каждой строки преобразует её в AbilityType с помощью StringToAbilityType, создаёт способность с помощью AbilityFactory::createAbility. После чего добавляет созданную способность в очередь.

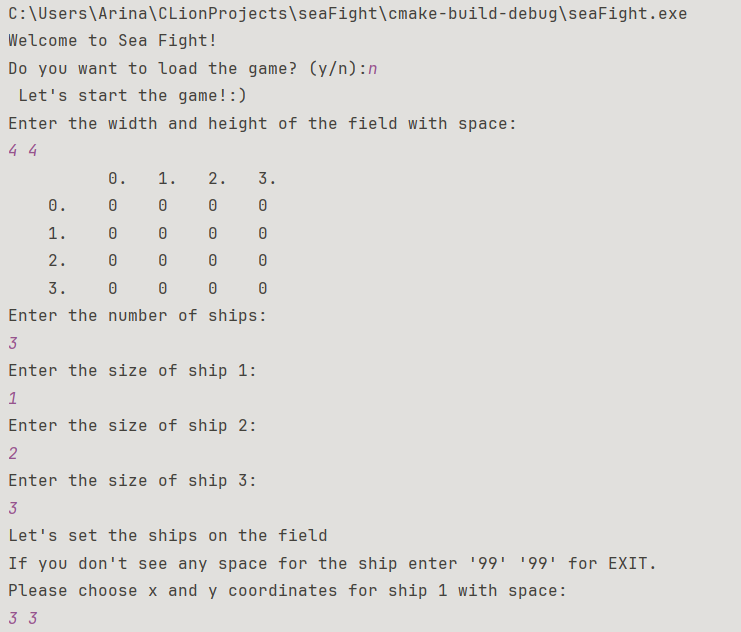
# UML – диаграмма классов

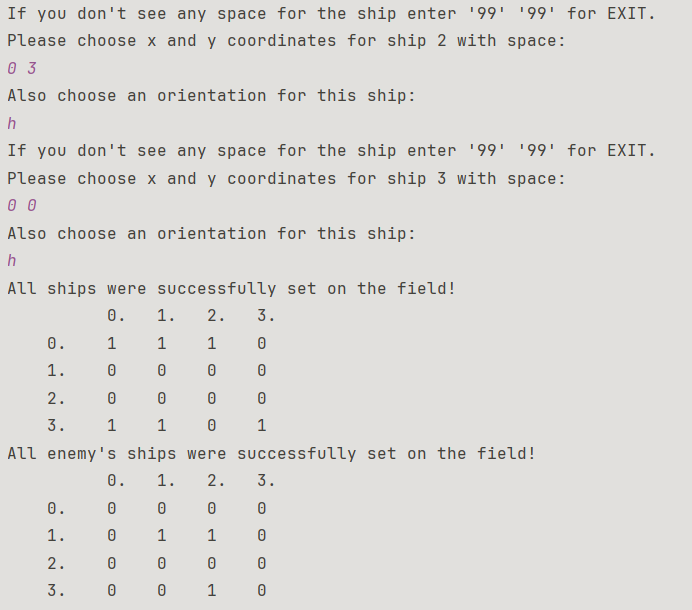
****

**Тестирование**

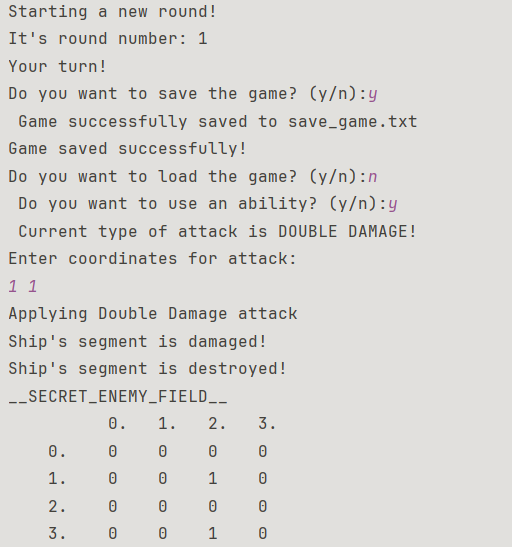
Демонстрация начала игры. Установка пользователем размеров поля, количества и размеров кораблей для их размещения на поле с

определенными координатами и ориентацией. А также автоматическая расстановка кораблей на поле противника.

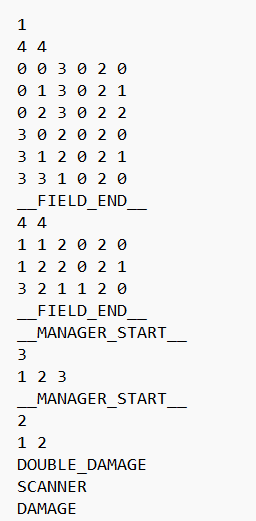




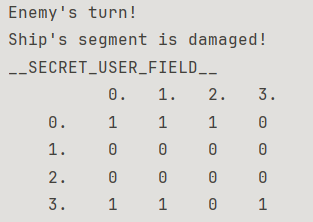
Начало нового раунда и ход пользователя.



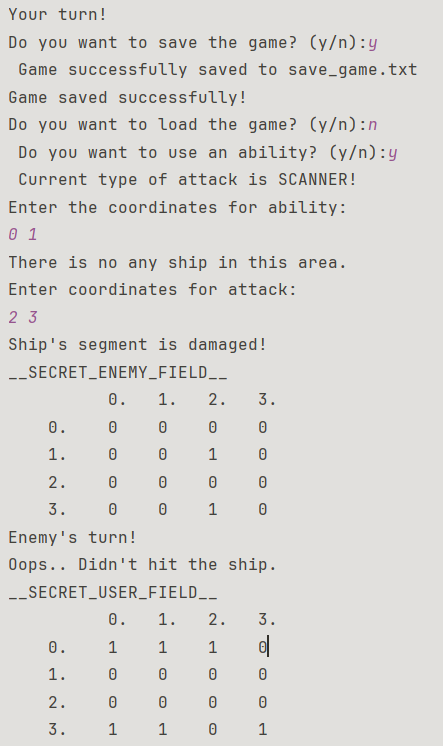
Содержимое файла save\_game.txt при сохранении.



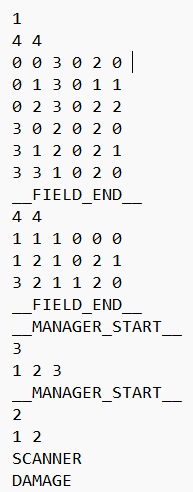
Ход противника.



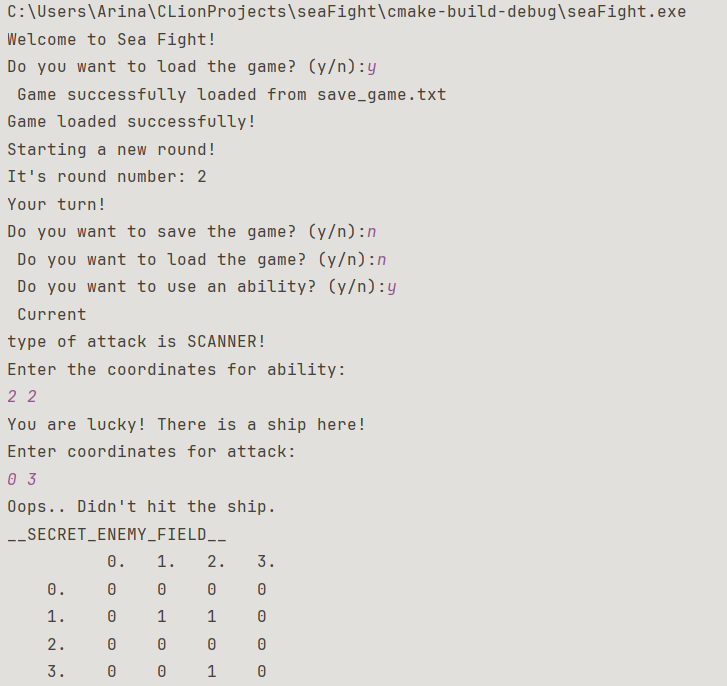
Ход пользователя. Повторное сохранение игры после применения первой способности. Использование второй способности.



Содержимое файла save\_game.txt после второго сохранения.



Демонстрация загрузки игры из файла при запуске программы (аналогичную загрузку возможно производить при любом ходе пользователя).



# Выводы

В ходе выполнения лабораторной работы был создан класс игры (Game), класс состояния игры, класс обёртки для работы с файлом, обстрела, класс сохранения поля, класс сохранения очереди способностей, класс сохранения менеджера кораблей. Благодаря данным классам был реализован полноценный процесс сохранения и загрузки состояния игры, включая игровые поля, менеджеров и способности.