**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. 3388 |  | Шубин П.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

## Создать класс игры, который реализует игровой цикл с чередующимися ходами игрока и компьютерного противника. Игрок может применить способность и атаковать, а враг — только атаковать. При поражении игрока начинается новая игра, при победе — новый раунд, при этом сохраняются состояние поля и способности игрока. Класс должен содержать методы для управления игрой, начала новой игры и выполнения хода. Необходимо также разработать класс состояния игры с переопределенными операторами ввода и вывода для работы с потоками, а также функции для сохранения и загрузки игры, обеспечивающие возможность продолжить игру после перезапуска программы. Игровые сущности не должны зависеть от состояния игры, а управление игровым процессом должно осуществляться через обертки над командами. Для работы с файлами следует использовать идиому RAII для корректного управления ресурсами.

## Задание

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 Game

Класс Game представляет собой основную игровую логику, управляя состоянием игры, ходами пользователя и противника, а также возможностью сохранения и загрузки игры. Этот класс взаимодействует с объектами GameState и ShipManager для управления полями игроков и обработкой различных игровых событий.

Поля класса Game:

* GameState gameState — объект, который хранит текущее состояние игры, включая поля игроков, раунд и информацию о ходах.
* ShipManager& userManager — ссылка на менеджер кораблей пользователя, который управляет расположением и состоянием кораблей игрока.
* ShipManager& enemyManager — ссылка на менеджер кораблей врага, который управляет кораблями противника.
* bool isRunning — флаг, указывающий, запущена ли игра.

Методы класса Game:

* Game(ShipManager& userManager, ShipManager& enemyManager) — конструктор, инициализирующий игру с заданными менеджерами кораблей для пользователя и врага, а также создающий объект состояния игры.
* void startNewGame() — запускает новую игру, сбрасывая состояние и инициализируя все необходимые объекты.
* void endGame() — завершает игру, проверяя, выиграл ли пользователь или проиграл, и в зависимости от результата либо начинается новый раунд, либо игра завершена.
* void startNewRound() — начинает новый раунд, сбрасывая состояние поля и увеличивая номер раунда.
* void makeMove() — выполняет ход, в зависимости от того, чей сейчас ход — пользователя или врага.
* void playerTurn() — логика хода пользователя, включая выбор способности и атаку по координатам.
* void enemyTurn() — логика хода врага, который атакует пользователя.
* void saveGame(const std::string& filename) — сохраняет текущее состояние игры в файл.
* void loadGame(const std::string& filename) — загружает сохраненную игру из файла.
* bool isGameRunning() const — возвращает состояние игры (идет ли она).
* const GameState& getGameState() const — возвращает текущее состояние игры.
* void displayGameState() const — выводит текущее состояние игры, включая поля игроков и номер раунда.
* void run() — основной игровой цикл, который запускает игру, чередует ходы и предлагает сохранить игру после каждого раунда.

Класс GameState

Класс GameState представляет собой основное состояние игры, включая игровые поля, способности, текущий раунд и информацию о ходе игры. Он управляет состоянием поля для пользователя и противника, а также координирует применение способностей и атаки. Этот класс также отвечает за сохранение и загрузку состояния игры, что позволяет игроку продолжать с того места, где он остановился.

Поля класса GameState:

* GameField userField — объект, представляющий игровое поле пользователя.
* GameField enemyField — объект, представляющий игровое поле противника.
* AbilityManager abilityManager — менеджер способностей, который управляет применением способностей игрока.
* ShipManager& userManager — ссылка на менеджер кораблей пользователя.
* ShipManager& enemyManager — ссылка на менеджер кораблей противника.
* bool isUserTurn — флаг, указывающий, чей сейчас ход: пользователя или противника.
* bool gameOver — флаг, указывающий, завершена ли игра.
* int roundNumber — текущий номер раунда.

Методы класса GameState:

* GameState(int fieldWidth, int fieldHeight, ShipManager& userManager, ShipManager& enemyManager) — конструктор, инициализирующий игровые поля и менеджеры кораблей для пользователя и противника.
* void reset(ShipManager& userManager, ShipManager& enemyManager) — сбрасывает состояние игры, включая поля, способности и переменные состояния.
* void toggleTurn() — переключает ход между пользователем и противником.
* void applyAbility() — применяет способность пользователя, запрашивая у него координаты.
* void enemyAttack() — генерирует случайную атаку от имени врага, нанося ущерб полю пользователя.
* void attackEnemy(int x, int y) — выполняет атаку на поле врага по заданным координатам.
* void saveToFile(const std::string& filename) const — сохраняет текущее состояние игры в файл.
* static GameState loadFromFile(const std::string& filename, ShipManager& userManager, ShipManager& enemyManager) — загружает сохраненное состояние игры из файла.
* void setGameOver(bool isGameOver) — устанавливает состояние игры как завершенное или продолжающееся.
* bool isGameOver() const — возвращает состояние игры (завершена ли игра).
* bool getUserTurn() const — возвращает, чей сейчас ход.
* int getRoundNumber() const — возвращает номер текущего раунда.
* const GameField& getUserField() const — возвращает поле пользователя.
* const GameField& getEnemyField() const — возвращает поле врага.
* void incrementRound() — увеличивает номер раунда.
* void placeUserShips() — позволяет пользователю размещать свои корабли на поле.
* void placeEnemyShips() — размещает корабли противника случайным образом.

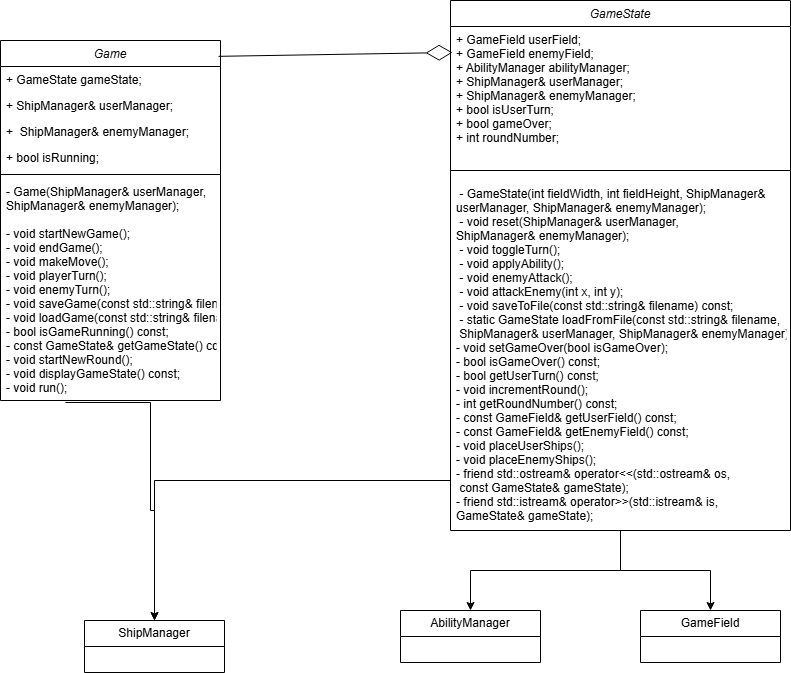
Операторы:

* + - std::ostream& operator<<(std::ostream& os, const GameState& gameState) — перегрузка оператора вывода, позволяющая вывести информацию о состоянии игры на экран.
* std::istream& operator>>(std::istream& is, GameState& gameState) — перегрузка оператора ввода, позволяющая загрузить состояние игры из потока.

**main()**

Функция main() реализует основную программу для игры "Морской бой", которая позволяет пользователю начать новую игру, загрузить сохраненную игру или выйти. В нем создаются объекты для управления кораблями пользователя и противника, а также объект игры, который управляет процессом. В меню пользователю предлагаются три опции: начать новую игру, загрузить игру или выйти. При начале новой игры запускается игровой цикл, в котором игрок и противник поочередно делают ходы, а после каждого хода можно сохранить игру. Если пользователь выбирает загрузку игры, программа восстанавливает состояние из файла и продолжает игру с того места, где она была прервана.

## UML-диаграмма классов



## 

## Выводы

В ходе выполнения задания был создан класс игры, реализующий полноценный игровой цикл с чередованием ходов пользователя и компьютерного врага. Пользователь может применить способности и атаковать, а компьютерный противник только атакует. В случае поражения пользователя игра начинается заново, а при победе — начинается новый раунд, при этом сохраняются состояния поля и способности пользователя. Реализованы методы для управления игрой, начала новой игры и выполнения хода. Также разработан класс состояния игры с переопределением операторов ввода и вывода для работы с потоками данных, а также функции для сохранения и загрузки игры, что позволяет продолжать игровой процесс после перезапуска программы. Игровые сущности независимы от состояния игры, а управление процессом осуществляется через обертки над командами, что соответствует требованиям задания.