**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. 3343 |  | Малиновский А.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать класс игры, реализующий игровой цикл, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполнить атаку, а компьютерный враг только наносит атаку. Реализовать сохранение и загрузку игры, а также обработку исключительных ситуаций для обеспечения корректного функционирования 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.

## **Выполнение работы**

Для работы с файлом создан класс обёртки FileWrapper, который соответствует идиоме RAII. Классы Serializer и Desializer созданы для преобразования сложных данных в формат json. Реализован класс Game содержащий своё состояние GameState, а также классы User и Bot, которые оборачивают поле, менеджер кораблей, иенеджер способностей для юзера. При желании пользователя загрузиться или сохраниться вызывается соответствующий метод класса Game в котором создаётся GameSaver или GameLoader принимающие в конструкторе GameState. GameSaver с помощью перегруженного оператора << записывает в файл состояние игры, если файл не открывается выбрасывается исключение, также данные хэшируются для проверки на валидность файла. GameLoader с помощью оператора >> считывает из файла данные, потом проверяет валидность данных, в случае если файл был изменён вручную или файл был повреждён выбрасываются исключения. При загрузке Game в методе load заменяет свои поля игрока и бота на поля полученные из изменённого GameState. Также реализованы возможности чередовать ходы игрока и бота, игрок может использовать способность и атаковать, бот только атаковать, при уничтожении всех кораблей бота, создаётся новый бот, а раунд увеличивается на 1, при проигрыше игрока раунд становится равным 1, а игрок создаётся заново.

**Подробное описание реализованных классов**

Класс Deserializer предназначен для десериализации объектов из JSON-представления. Он использует библиотеку nlohmann::json для работы с JSON и предоставляет методы для десериализации различных типов данных, таких как ShipManager, GameField, AbilityManager и примитивные типы.

Поля класса:

* nlohmann::json& j: Ссылка на JSON-объект, из которого будет производиться десериализация.

Методы класса:

1. Конструктор Deserializer(nlohmann::json& j):

Инициализирует объект Deserializer, принимая ссылку на JSON-объект, из которого будет производиться десериализация.

1. Метод void from\_json(ShipManager& shipManager, std::string key):

Десериализует объект типа ShipManager из JSON-объекта по указанному ключу.

1. Метод void from\_json(GameField& field, std::string key):

Десериализует объект типа GameField из JSON-объекта по указанному ключу.

1. Метод void from\_json(AbilityManager& abilityManager, std::string key):

Десериализует объект типа AbilityManager из JSON-объекта по указанному ключу.

1. Метод void from\_json(int& number, std::string key):

Десериализует целое число из JSON-объекта по указанному ключу.

1. Метод bool checkIsValid():

Проверяет, является ли JSON-объект валидным для десериализации. Возвращает true, если JSON-объект валиден, и false в противном случае.

Класс Serializer предназначен для сериализации объектов в JSON-представление. Он использует библиотеку nlohmann::json для работы с JSON и предоставляет методы для сериализации различных типов данных, таких как ShipManager, GameField, AbilityManager и примитивные типы.

Поля класса:

* nlohmann::json& j: Ссылка на JSON-объект, в который будет производиться сериализация. Это основной источник данных для сериализации.

Методы класса:

1. Конструктор Serializer(nlohmann::json& j):

Инициализирует объект Serializer, принимая ссылку на JSON-объект, в который будет производиться сериализация.

1. Метод void to\_json(ShipManager& shipManager, std::string key):

Сериализует объект типа ShipManager в JSON-объект по указанному ключу.

1. Метод void to\_json(GameField& field, std::string key):

Сериализует объект типа GameField в JSON-объект по указанному ключу.

1. Метод void to\_json(AbilityManager& abilityManager, std::string key):

Сериализует объект типа AbilityManager в JSON-объект по указанному ключу.

1. Метод void to\_json(int number, std::string key):

: Сериализует целое число в JSON-объект по указанному ключу.

1. Метод void hashFile():

: Вычисляет хеш-сумму файла, чтобы обеспечить целостность данных. Этот метод может использоваться для проверки, не был ли файл изменен после сериализации.

Класс Game представляет собой основной класс игры, который управляет игровым процессом, включая чередование ходов пользователя и компьютерного врага, сохранение и загрузку игры, а также управление состоянием игры.

Поля класса:

* std::shared\_ptr<User> user: Умный указатель на объект игрока.
* std::shared\_ptr<Bot> bot: Умный указатель на объект компьютерного противника.
* GameState state: Объект состояния игры, содержащий информацию о текущем раунде, игроке и боте.
* ConsoleDisplayer displayer: Объект для отображения информации на консоли.
* InputHandler inputHandler: Объект для обработки ввода пользователя.
* Randomizer randomizer: Объект для генерации случайных чисел.
* bool isUserTurn: Флаг, указывающий, чей ход (пользователя или бота).
* int currentRound: Переменная, хранящая текущий раунд игры.

Методы класса:

1. Конструктор Game():

Инициализирует объект Game, устанавливая начальные значения для игрока, бота, состояния игры и других полей.

1. Метод void saveGame(const std::string& filename):

Сохраняет текущее состояние игры в файл с указанным именем. Использует объект GameSaver для сохранения состояния игры.

1. Метод void loadGame(const std::string& filename):

Загружает состояние игры из файла с указанным именем. Использует объект GameLoader для загрузки состояния игры.

1. Метод void makeUserTurn():

Выполняет ход пользователя. Пользователь может выбрать способность или атаку. В случае успешного применения способности или атаки, обновляет состояние игры.

1. Метод void makeBotTurn():

Выполняет ход компьютерного противника. Бот выбирает случайную координату и атакует её.

1. Метод void makeTurn():

Управляет чередованием ходов пользователя и бота. Отображает текущее состояние игры и вызывает соответствующий метод для выполнения хода.

1. Метод void startNewGame():

Начинает новую игру, сбрасывая текущий раунд и создавая новые объекты для игрока.

1. Метод void startNewRound():

Начинает новый раунд игры, увеличивая текущий раунд и создавая новый объект для бота.

1. Метод bool isOver():

Проверяет, закончена ли игра. Возвращает true, если все корабли игрока уничтожены, и false в противном случае.

Класс FileWrapper предназначен для работы с файлами, обеспечивая безопасное открытие, чтение и запись данных в формате JSON. Он использует библиотеку nlohmann::json для работы с JSON и предоставляет методы для чтения и записи JSON-объектов в файл.

Поля класса:

* mutable std::fstream file: Объект потока файла, который используется для чтения и записи данных. Ключевое слово mutable позволяет изменять это поле даже в константных методах.

Методы класса:

1. Конструктор FileWrapper(const std::string& filename, std::ios::openmode mode):

Инициализирует объект FileWrapper, открывая файл с указанным именем и режимом доступа. Если файл не удается открыть, выбрасывается исключение FileOpeningException.

1. Деструктор ~FileWrapper():

Закрывает файл, если он был открыт.

1. Метод void write(nlohmann::json& j):

Записывает JSON-объект в файл. Если файл не открыт или находится в некорректном состоянии, выбрасывается исключение FileOpeningException.

1. Метод void read(nlohmann::json& j):

Читает JSON-объект из файла. Если файл не открыт или находится в некорректном состоянии, выбрасывается исключение FileOpeningException.

Структуры User и Bot предназначены для представления игрока и компьютерного противника в игре. Обе структуры содержат информацию о игровом поле, менеджере кораблей и, в случае игрока, менеджере способностей.

Поля структуры User:

* GameField field: Игровое поле игрока.
* ShipManager shipManager: Менеджер кораблей игрока.
* AbilityManager abilityManager: Менеджер способностей игрока.
* Randomizer randomizer: Генератор случайных чисел, используемый для различных случайных операций.

Методы структуры User:

1. Конструктор User(GameField& field, AbilityManager& abilityManager, ShipManager& sm):

Инициализирует объект User, принимая ссылки на игровое поле, менеджер способностей и менеджер кораблей.

1. Конструктор User():

Конструктор по умолчанию для объекта User.

Поля структуры Bot:

* GameField field: Игровое поле компьютерного противника.
* ShipManager shipManager: Менеджер кораблей компьютерного противника.
* Randomizer randomizer: Генератор случайных чисел, используемый для различных случайных операций.

Методы структуры Bot:

1. Конструктор Bot(GameField& field, ShipManager& sm):

Инициализирует объект Bot, принимая ссылки на игровое поле и менеджер кораблей.

1. Конструктор Bot():

: Конструктор по умолчанию для объекта Bot.

Класс GameState предназначен для управления состоянием игры, включая информацию о текущем раунде, игроке и компьютерном противнике. Он предоставляет методы для получения и установки текущего раунда, а также для доступа к объектам игрока и бота.

Поля класса:

* std::shared\_ptr<User> user: Умный указатель на объект игрока.
* std::shared\_ptr<Bot> bot: Умный указатель на объект компьютерного противника.
* int currentRound: Переменная, хранящая текущий раунд игры.

Методы класса:

1. Конструктор GameState(std::shared\_ptr<User>user, std::shared\_ptr<Bot> bot, int& current\_round):

Инициализирует объект GameState, принимая умные указатели на объекты игрока и бота, а также ссылку на переменную, хранящую текущий раунд.

1. Метод std::shared\_ptr<User> getUser():

Возвращает умный указатель на объект игрока.

1. Метод std::shared\_ptr<Bot> getBot():

Возвращает умный указатель на объект компьютерного противника.

1. Метод int getCurrentRound():

Возвращает текущий раунд игры.

1. Метод void setCurrentRound(int r):

Устанавливает текущий раунд игры.

Класс GameSaver предназначен для сохранения состояния игры в файл. Он использует объект GameState для получения текущего состояния игры и сохраняет его в формате JSON с помощью объекта FileWrapper.

Поля класса:

* GameState& state: Ссылка на объект GameState, содержащий текущее состояние игры.

Методы класса:

1. Конструктор GameSaver(GameState& state):

Инициализирует объект GameSaver, принимая ссылку на объект GameState.

1. Метод void save(const std::string& filename):

Сохраняет текущее состояние игры в файл с указанным именем. Использует объект FileWrapper для записи данных в файл. Если файл не удается открыть, выбрасывается исключение FileOpeningException.

Перегруженный оператор operator<< предназначен для записи состояния игры в объект FileWrapper. Он сериализует состояние игры в JSON-объект и записывает его в файл.

Класс GameLoader предназначен для загрузки состояния игры из файла. Он использует объект GameState для восстановления состояния игры из формата JSON с помощью объекта FileWrapper.

Поля класса:

* GameState& state: Ссылка на объект GameState, в который будет загружено состояние игры.

Методы класса:

1. Конструктор GameLoader(GameState& state):

Инициализирует объект GameLoader, принимая ссылку на объект GameState.

1. Метод void load(const std::string& filename):

Загружает состояние игры из файла с указанным именем. Использует объект FileWrapper для чтения данных из файла. Если файл не удается открыть, выбрасывается исключение FileOpeningException. После загрузки данных вызывается метод placeShipsBack для восстановления позиций кораблей на игровом поле.

1. Метод void placeShipsBack(ShipManager& shipManager, GameField& field):

Восстанавливает позиции кораблей на игровом поле на основе данных из ShipManager. Устанавливает ссылки на корабли в соответствующих ячейках игрового поля.

Перегруженный оператор operator>> предназначен для чтения состояния игры из объекта FileWrapper. Он десериализует состояние игры из JSON-объекта и восстанавливает его в объекте GameState.

Класс FileOpeningException предназначен для обработки исключений, возникающих при открытии файла. Он наследуется от стандартного класса std::exception и переопределяет метод what(), чтобы предоставить сообщение об ошибке.

Поля класса:

* std::string message: Строка, содержащая сообщение об ошибке.

Методы класса:

1. Конструктор FileOpeningException():

Инициализирует объект FileOpeningException, устанавливая сообщение об ошибке.

1. Метод virtual const char\* what() const noexcept override:

Возвращает сообщение об ошибке в виде строки C-style. Переопределяет метод what() из базового класса std::exception.

Класс FileWasDamagedException предназначен для обработки исключений, возникающих при обнаружении поврежденного файла. Он также наследуется от стандартного класса std::exception и переопределяет метод what(), чтобы предоставить сообщение об ошибке.

Поля класса:

* std::string message: Строка, содержащая сообщение об ошибке.

Методы класса:

1. Конструктор FileWasDamagedException():
   * Инициализирует объект FileWasDamagedException, устанавливая сообщение об ошибке.
2. Метод virtual const char\* what() const noexcept override:

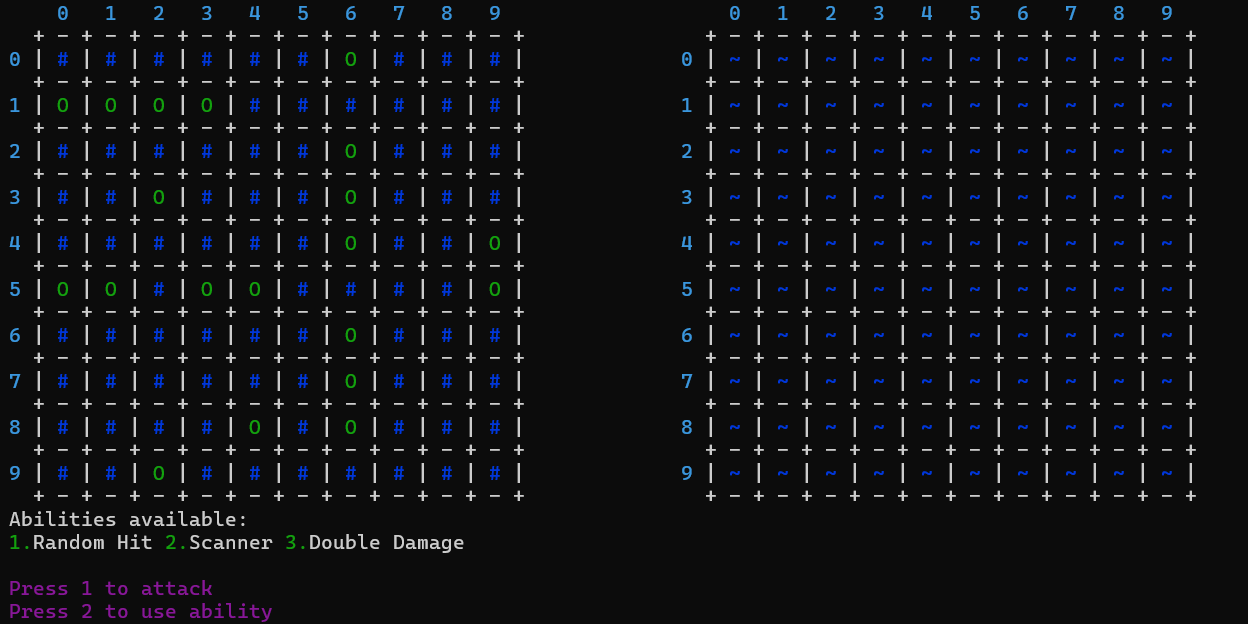
Возвращает сообщение об ошибке в виде строки C-style. Переопределяет метод what() из базового класса std::exception.

Разработанный программный код см. в приложении А.

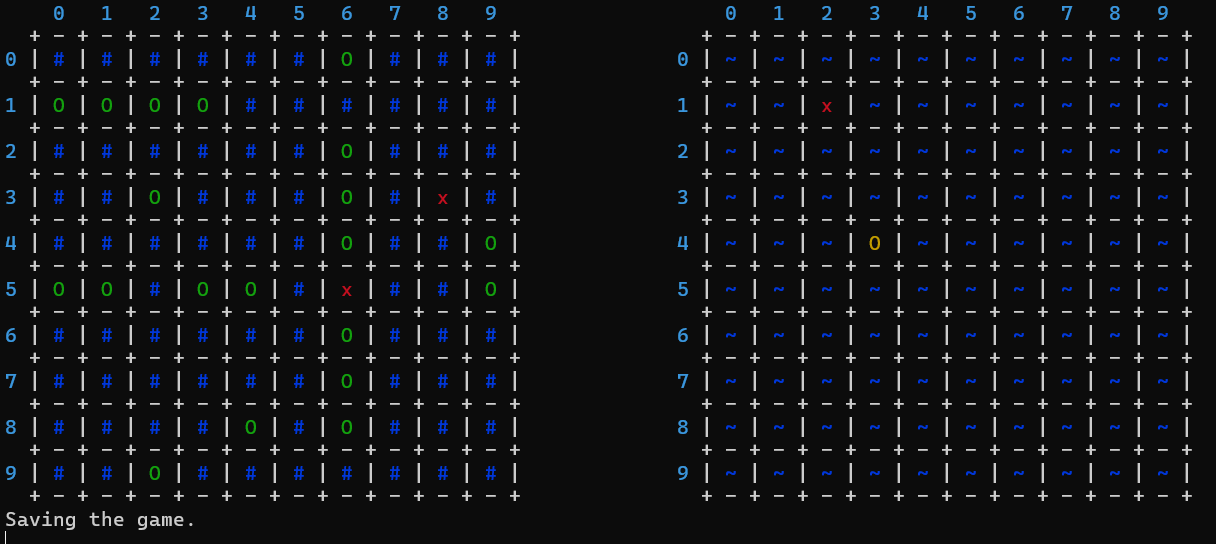
## **Тестирование**

Реализован игровой цикл для проверки написанных методов.

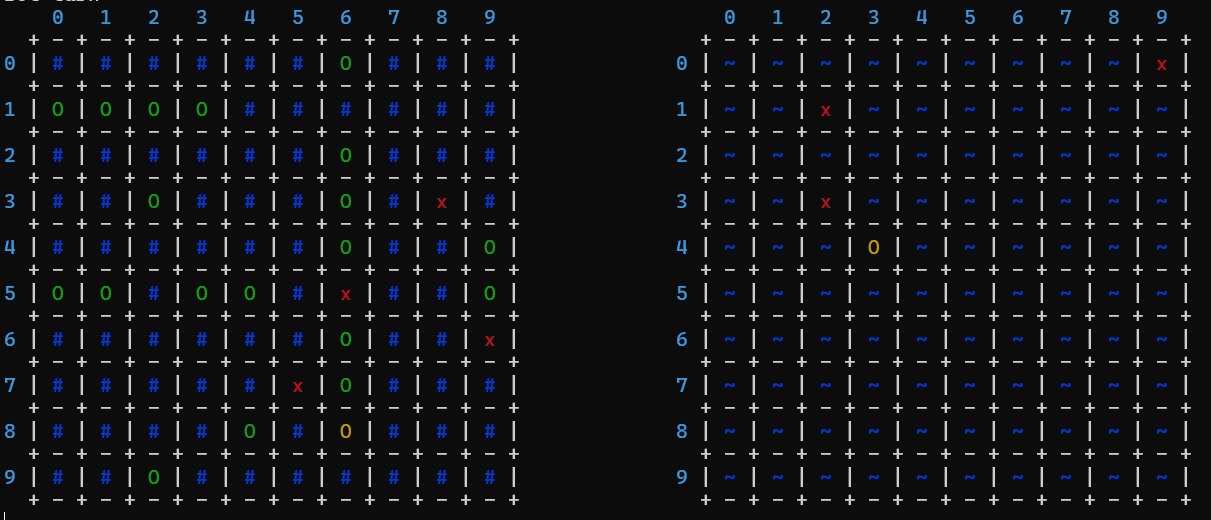
Вывод программы при ходе игрока:



Сохранение



Продолжение игры:



Загрузка

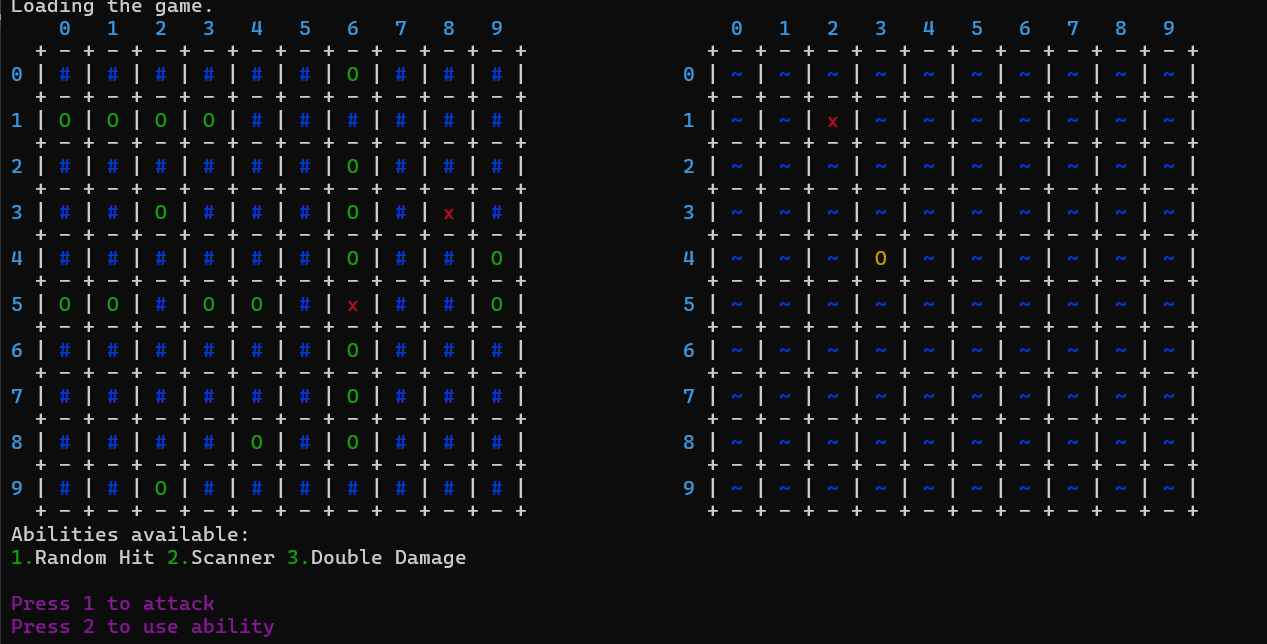
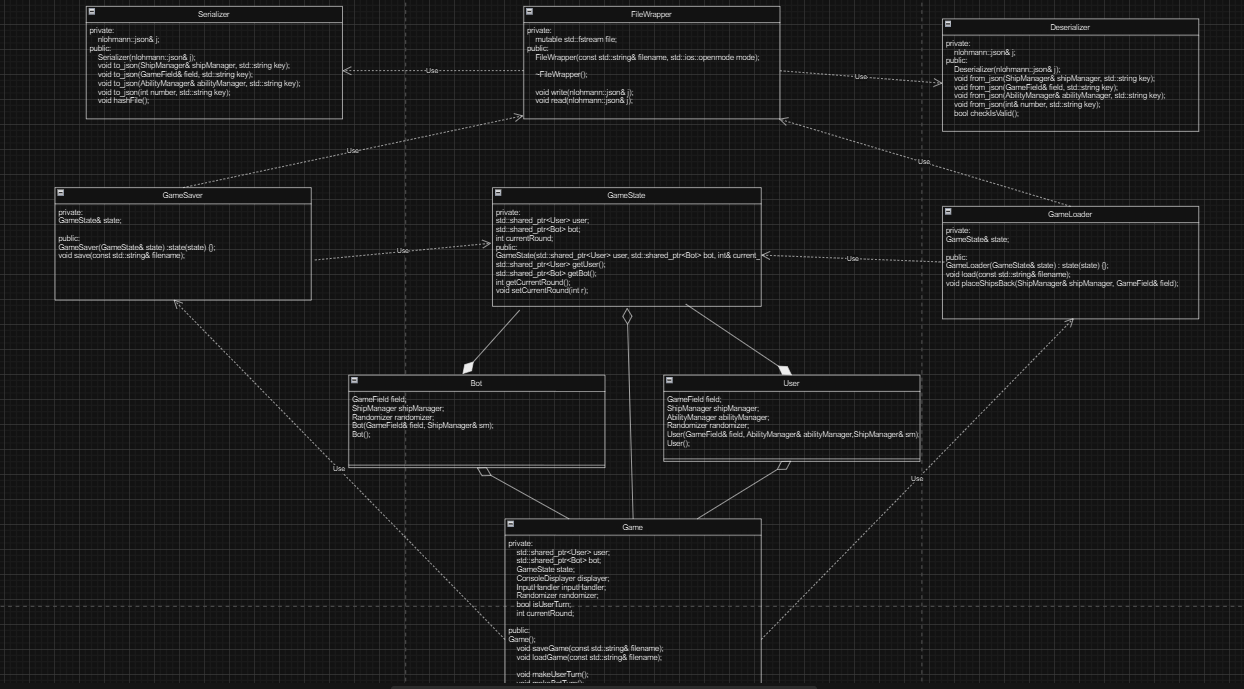


Диаграмма классов:



**Выводы**

В работе над классом игры была реализована система чередования ходов пользователя и компьютерного врага, где игрок может применять способности и атаковать, а враг только атакует. Реализация класса состояния игры позволяет эффективно управлять текущим состоянием игры, включая сохранение и загрузку игры в любой момент, когда у пользователя приоритет в игре. Использование идиомы RAII при работе с файлами обеспечивает безопасность и корректное управление ресурсами.

# **Приложение А Исходный код программы**

Название файла: Game.cpp

#include "Game.hpp"

Game::Game() : displayer(ConsoleDisplayer()), inputHandler(InputHandler()), randomizer(Randomizer()) {

isUserTurn = true;

currentRound = 1;

user = std::make\_shared<User>();

bot = std::make\_shared<Bot>();

state= new GameState(user, bot, currentRound);

}

void Game::makeUserTurn() {

Coordinates userCoords{ -1,-1 };

int currentDamage = 1;

UserParams params(bot->field, bot->shipManager, userCoords,currentDamage);

bool abilitySuccess = false;

displayer.displayAttackOrAbilityChoice();

int choice = inputHandler.handleAttackorAbility();

while (choice==2 && !abilitySuccess) {

try {

if (user->abilityManager.getAbilityCreator(0).getIsCoordsRequired()) {

displayer.displayWaitingCoordinatesInput();

params.coords = inputHandler.handleCoordsInput();

}

AbilityResult result = user->abilityManager.useAbility(params);

switch (result)

{

case AbilityResult::ShipDestroyed:

user->abilityManager.addRandomAbilityCreator();

break;

case AbilityResult::SegmentDetected:

displayer.displaySegmentWasFoundMessage();

break;

case AbilityResult::SegmentNotFound:

displayer.displaySegmentWasNotFoundMessage();

break;

default:

break;

}

user->abilityManager.popAbilityCreator();

abilitySuccess = true;

}

catch (NoAbilitiesException& e) {

displayer.displayException(e);

break;

}

catch (OutOfBoundsException& e) {

displayer.displayException(e);

}

}

displayer.displayWaitingCoordinatesInput();

bool attackSuccess = false;

while (!attackSuccess) {

try {

params.coords = inputHandler.handleCoordsInput();

for (int i = 0; i < params.currentDamage; i++)

{

if (params.field.attackCell(params.coords)) {

user->abilityManager.addRandomAbilityCreator();

}

}

params.currentDamage = 1;

attackSuccess = true;

}

catch (OutOfBoundsException& e) {

displayer.displayException(e);

displayer.displayWaitingCoordinatesInput();

}

}

}

void Game::makeBotTurn() {

Bot bot(user->field, user->shipManager);

bool attackSuccess = false;

while (!attackSuccess) {

Coordinates coords = randomizer.getRandomCoordinates();

auto& cell = user->field.getFieldCell(coords);

if (cell.value != CellValue::Miss || (cell.value == CellValue::ShipSegment

&& cell.ship->getSegment(cell.segmentIndex)->status != SegmentStatus::DESTROYED)) {

user->field.attackCell(coords);

attackSuccess = true;

}

}

}

void Game::makeTurn() {

if (isUserTurn)

{

displayer.displayTwoFields(user->field, bot->field);

displayer.displayAbilities(user->abilityManager);

makeUserTurn();

isUserTurn = false;

displayer.displayTwoFields(user->field,bot->field);

displayer.displayString("Press 'q' to quit, 's' to save game, 'l' to load game,'p' to play");

}

else {

displayer.displayString("Bot turn");

makeBotTurn();

displayer.displayTwoFields(user->field, bot->field);

isUserTurn = true;

}

}

void Game::startNewGame() {

currentRound = 1;

user = std::make\_shared<User>();

}

void Game::startNewRound() {

currentRound += 1;

bot = std::make\_shared<Bot>();

}

void Game::saveGame(const std::string& filename) {

GameSaver saver(\*this->state);

try {

saver.save(filename);

}

catch (FileOpeningException& e) {

displayer.displayException(e);

return;

}

}

void Game::loadGame(const std::string& filename) {

GameLoader loader(\*this->state);

try {

loader.load(filename);

}

catch (FileOpeningException& e) {

displayer.displayException(e);

return;

}

catch (FileWasDamagedException& e) {

displayer.displayException(e);

return;

}

catch (nlohmann::json::parse\_error& e) {

displayer.displayException(e);

return;

}

isUserTurn = true;

user = state->getUser();

bot = state->getBot();

}

bool Game::isOver() {

return user->shipManager.allShipsDestroyed();

}

Название файла: FileWrapper.cpp

#include "FileWrapper.hpp"

FileWrapper::FileWrapper(const std::string& filename, std::ios::openmode mode) : file(filename,mode) {

if (!file) {

throw FileOpeningException();

};

};

FileWrapper::~FileWrapper() {

if (file.is\_open())

file.close();

}

void FileWrapper::write(nlohmann::json& j) {

if (!file.is\_open() || !file.good()) {

throw FileOpeningException();

}

file << j.dump(4);

}

void FileWrapper::read(nlohmann::json& j) {

if (!file.is\_open() || !file.good()) {

throw FileOpeningException();

}

file >> j;

}

}} Название файла: GameSaver.cpp

# #include "GameSaver.hpp"

FileWrapper& operator<<(FileWrapper& file, GameState& state) {

nlohmann::json j;

Serializer serializer(j);

serializer.to\_json(state.getUser()->shipManager, "userShipManager");

serializer.to\_json(state.getUser()->field, "userField");

serializer.to\_json(state.getUser()->abilityManager, "abilityManager");

serializer.to\_json(state.getBot()->shipManager, "botShipManager");

serializer.to\_json(state.getBot()->field, "botField");

serializer.to\_json(state.getCurrentRound(), "currentRound");

serializer.hashFile();

file.write(j);

return file;

}

void GameSaver::save(const std::string& filename) {

try {

FileWrapper file(filename,std::ios::out | std::ios::trunc);

file << state;

}

catch (FileOpeningException& e) {

throw e;

}

}}

} Название файла: GameLoader.cpp

#include "GameLoader.hpp"

FileWrapper& operator>>(FileWrapper& file, GameState& state) {

nlohmann::json j;

file.read(j);

Deserializer deserializer(j);

if (!deserializer.checkIsValid()) {

throw(FileWasDamagedException());

}

ShipManager sm;

ShipManager sm2;

GameField gf;

GameField gf2;

AbilityManager abilityManager;

int currentRound;

deserializer.from\_json(sm, "userShipManager");

deserializer.from\_json(gf, "userField");

deserializer.from\_json(sm2, "botShipManager");

deserializer.from\_json(gf2, "botField");

deserializer.from\_json(abilityManager, "abilityManager");

deserializer.from\_json(currentRound, "currentRound");

state.getBot()->shipManager = sm2;

state.getBot()->field = gf2;

state.getUser()->shipManager = sm;

state.getUser()->field = gf;

state.getUser()->abilityManager = abilityManager;

state.setCurrentRound(currentRound);

return file;

}

void GameLoader::placeShipsBack(ShipManager& shipManager, GameField& field) {

for (size\_t i = 0; i < shipManager.getShipsAmount(); i++)

{

for (size\_t j = 0; j < shipManager.getShip(i).getLength(); j++)

{

Coordinates coords = shipManager.getShip(i).getSegment(j)->coord;

auto& cell = field.getFieldCell(coords);

cell.ship = &shipManager.getShip(i);

cell.value = CellValue::ShipSegment;

cell.segmentIndex = j;

}

}

}

void GameLoader::load(const std::string& filename) {

try {

FileWrapper file(filename,std::ios::in);

file >> state;

}

catch (FileOpeningException& e) {

throw e;

}

placeShipsBack(state.getBot()->shipManager, state.getBot()->field);

placeShipsBack(state.getUser()->shipManager, state.getUser()->field);

}

Название файла: GameState.cpp

#include "GameState.hpp"

GameState::GameState(std::shared\_ptr<User> user, std::shared\_ptr<Bot> bot, int& current\_round) :

user(user), bot(bot), currentRound(currentRound) {};

std::shared\_ptr<User> GameState::getUser() {

return user;

}

std::shared\_ptr<Bot> GameState::getBot() {

return bot;

}

int GameState::getCurrentRound() {

return currentRound;

}

void GameState::setCurrentRound(int r) {

currentRound = r;

}

} Название файла: Serializer.cpp

#include "Serializer.hpp"

Serializer::Serializer(nlohmann::json& j) : j(j) {};

void Serializer::to\_json(ShipManager& shipManager, std::string key) {

nlohmann::json jsonShipManager = nlohmann::json::array();

for (int i = 0; i < shipManager.getShipsAmount(); i++) {

Ship& ship = shipManager.getShip(i);

// std::string shipKey = "ship" + std::to\_string(i);

nlohmann::json shipJson = {

{"length", ship.getLength()},

{"isPlaced", ship.getIsPlaced()},

{"isVertical", ship.getIsVertical()},

{"isDestroyed", ship.getIsDestroyed()},

{"segments", nlohmann::json::array()}

};

for (int jj = 0; jj < ship.getLength(); jj++) {

shipJson["segments"].push\_back({

{"status", ship.getSegment(jj)->status},

{"x", ship.getSegment(jj)->coord.x},

{"y", ship.getSegment(jj)->coord.y}

});

}

jsonShipManager.push\_back(shipJson);

}

j[key] = jsonShipManager;

}

void Serializer::to\_json(GameField& field, std::string key) {

nlohmann::json jsonField = nlohmann::json{};

jsonField["width"] = field.getWidth();

jsonField["height"] = field.getHeight();

jsonField["status"] = nlohmann::json::array();

jsonField["value"] = nlohmann::json::array();

for (int y = 0; y < field.getHeight(); y++)

{

for (int x = 0; x < field.getWidth(); x++)

{

auto& cell = field.getFieldCell(Coordinates{ x,y });

jsonField["status"].push\_back(cell.status);

jsonField["value"].push\_back(cell.value);

}

}

j[key] = jsonField;

}

void Serializer::to\_json(AbilityManager& abilityManager, std::string key){

nlohmann::json jsonAbilityManager = nlohmann::json{}.array();

AbilityPrinter printer;

for (size\_t i = 0; i < abilityManager.getAbilitiesSize(); i++)

{

jsonAbilityManager.push\_back(abilityManager.getAbilityCreator(i).accept(&printer));

}

j[key] = jsonAbilityManager;

}

void Serializer::to\_json(int number, std::string key) {

j[key] = number;

}

void Serializer::hashFile() {

std::string jsonStr = j.dump();

std::hash<std::string> hashFn;

j["checksum"] = hashFn(jsonStr);

} Название файла: Deserializer.cpp

#include "Deserializer.hpp"

Deserializer::Deserializer(nlohmann::json& j) : j(j) {};

void Deserializer::from\_json(ShipManager& shipManager, std::string key) {

const auto& jsonShipManager = j.at(key);

for (const auto& shipJson : jsonShipManager) {

Ship ship(shipJson["length"]);

ship.setIsPlaced(shipJson["isPlaced"]);

ship.setIsVertical(shipJson["isVertical"]);

ship.setIsDestroyed(shipJson["isDestroyed"]);

int i = 0;

for (const auto& segmentJson : shipJson["segments"]) {

auto segment=ship.getSegment(i);

segment->status = segmentJson["status"];

segment->coord.x = segmentJson["x"];

segment->coord.y = segmentJson["y"];

i++;

}

shipManager.addShip(ship);

}

}

void Deserializer::from\_json(GameField& field, std::string key) {

const auto& userField = j.at(key);

field = GameField(userField.at("width").get<int>(), userField.at("height").get<int>());

for (int y = 0; y < field.getHeight(); y++) {

for (int x = 0; x < field.getWidth(); x++) {

int index = y \* field.getWidth() + x;

auto& cell = field.getFieldCell(Coordinates{ x, y });

cell.status = userField.at("status").at(index).get<CellStatus>();

cell.value = userField.at("value").at(index).get<CellValue>();

}

}

}

void Deserializer::from\_json(AbilityManager& abilityManager, std::string key){

const auto& jsonAM = j.at(key);

for (const auto& abilityJson : jsonAM) {

if (abilityJson.get<std::string>() == "Random Hit") {

abilityManager.addAbilityCreator(new RandomHitCreator());

}

else if(abilityJson.get<std::string>() == "Double Damage"){

abilityManager.addAbilityCreator(new DoubleDamageCreator());

}

else if (abilityJson.get<std::string>() == "Scanner") {

abilityManager.addAbilityCreator(new ScannerCreator());

}

}

}

void Deserializer::from\_json(int& number, std::string key){

number = j[key];

}

bool Deserializer::checkIsValid() {

auto storedChecksum = j["checksum"];

j.erase("checksum");

std::string loadedJsonStr = j.dump();

std::hash<std::string> hashFn;

auto checksum = hashFn(loadedJsonStr);

if (checksum != storedChecksum) {

return false;

}

return true;

}

Название файла: Player.cpp

#ifndef PLAYER\_HPP

#define PLAYER\_HPP

#include "UserParams.hpp"

#include "Randomizer.hpp"

#include "ConsoleDisplayer.hpp"

#include "InputHandler.hpp"

struct User {

GameField field;

ShipManager shipManager;

AbilityManager abilityManager;

Randomizer randomizer;

User(GameField& field, AbilityManager& abilityManager,ShipManager& sm);

User();

};

struct Bot {

GameField field;

ShipManager shipManager;

Randomizer randomizer;

Bot(GameField& field, ShipManager& sm);

Bot();

};

#endif