**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 |  | Пименов П.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Связать классы, созданные в предыдущих лабораторных работах. Получить практический опыт по сериализации, десериализации объектов.

**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.

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.**

Задание успешно выполнено, необходимые классы созданы.

Описание реализованных классов:

* ActionType – перечисление, характеризующее тип действия, которое можно совершить в ходе игрока. Каждое действие может быть Once, Many или Terminating. От этой характеристики зависит возможность сделать данное действие и переход на следующий этап игры.
* Action – абстрактный класс Действия в ходе игрока. Содержит Тип действия, проверку возможности использовать действие, саму логику действия и эффект. Эффект – то, что может выполниться после использования действия (как, например, установка флага после использования Способности). Цель создания класса: попытаться максимально обобщить и унифицировать возможные действия в ходе игрока.
* ActionResult – небольшой интерфейс результата применения действия. Есть две реализации – ActionResultFailure и ActionResultSuccess. Создан с целью отделить слой вывода от класса Действия. Но поскольку в ходе действия получаются какие-то результаты, то их нужно как-то вернуть и, впоследствии, показать.
* AttackAction – действие Атака. Атакует вражеское поле по выбранным координатам.
* Deserializer – класс, отвечающий за десериализацию объектов. Создает/инициализирует экземпляры объектов по их JSON представлению.
* FileReader – класс, отвечающий за считывание сохранения из файла (и его загрузку в состояние игры).
* FileWriter – класс, отвечающий за сохранение состояния игры в файл.
* GameSession – класс, реализующий игровой цикл. Содержит методы инициализации, начала игры, перезапуска раунда, выполнения действия и др. Основной класс игры, который будет в дальнейшем расширен, однако для демонстрации работоспособности программы уже сейчас было необходимо реализовать некоторые компоненты, не входящие в задание лабораторной.
* GameState – класс состояния игры. Внутри себя содержит два указателя на игроков и оперирует над их состоянием (инициализирует / отдает на них ссылку). Также сделаны методы вывода класса состояния в поток и ввода из потока. Кроме того, класс используется в реализации класса Действия, поскольку имеет актуальную информацию о игроках.
* MoveAction – класс действия Движение. Передвигает выбранную клетку на указанную сдвиг.
* Player – класс игрока. По сути, это дата класс, содержащий экземпляр Поля, Менеджера кораблей, Менеджера способностей и т. д. Никакой логики, кроме хранения данных и выдачи ссылок на них, класс не имеет.
* Serializer – класс, отвечающий за сериализацию игровых сущностей. Формирует JSON представление сущности по ее экземпляру. JSON выбран по причине стабильного проверенного временем стандарта, удобного для чтения, парсинга, записи.
* UseAbilityAction – класс действия Использовать способность. Использует доступную способность у игрока.

Валидация сохранения происходит путем вычисления хэша строкового представления сериализованного состояния игры. Этот хэш также записывается в файл сохранения и используется для валидации. Если фактический хэш (вычисленный заново при загрузке) не совпадает с приписанным к сохранению – это приведет к исключению, которое будет обработано отдельно.

UML-диаграмму реализованных классов см. в приложении A.

Исходный код реализованных классов см. в приложении B.

**Тестирование.**

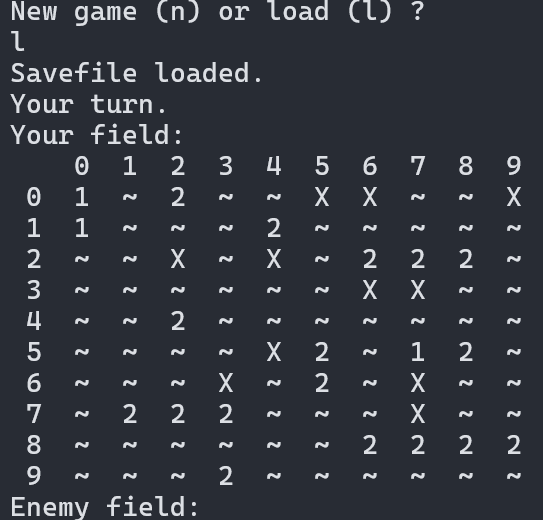


Рисунок 1 – загрузка сохранения



Рисунок 2 – запись сохранения

**Выводы.**

Классы, созданные в предыдущих лабораторных работах, связаны с помощью новых классов. Получен практический опыт по сериализации, десериализации объектов.

Приложение а

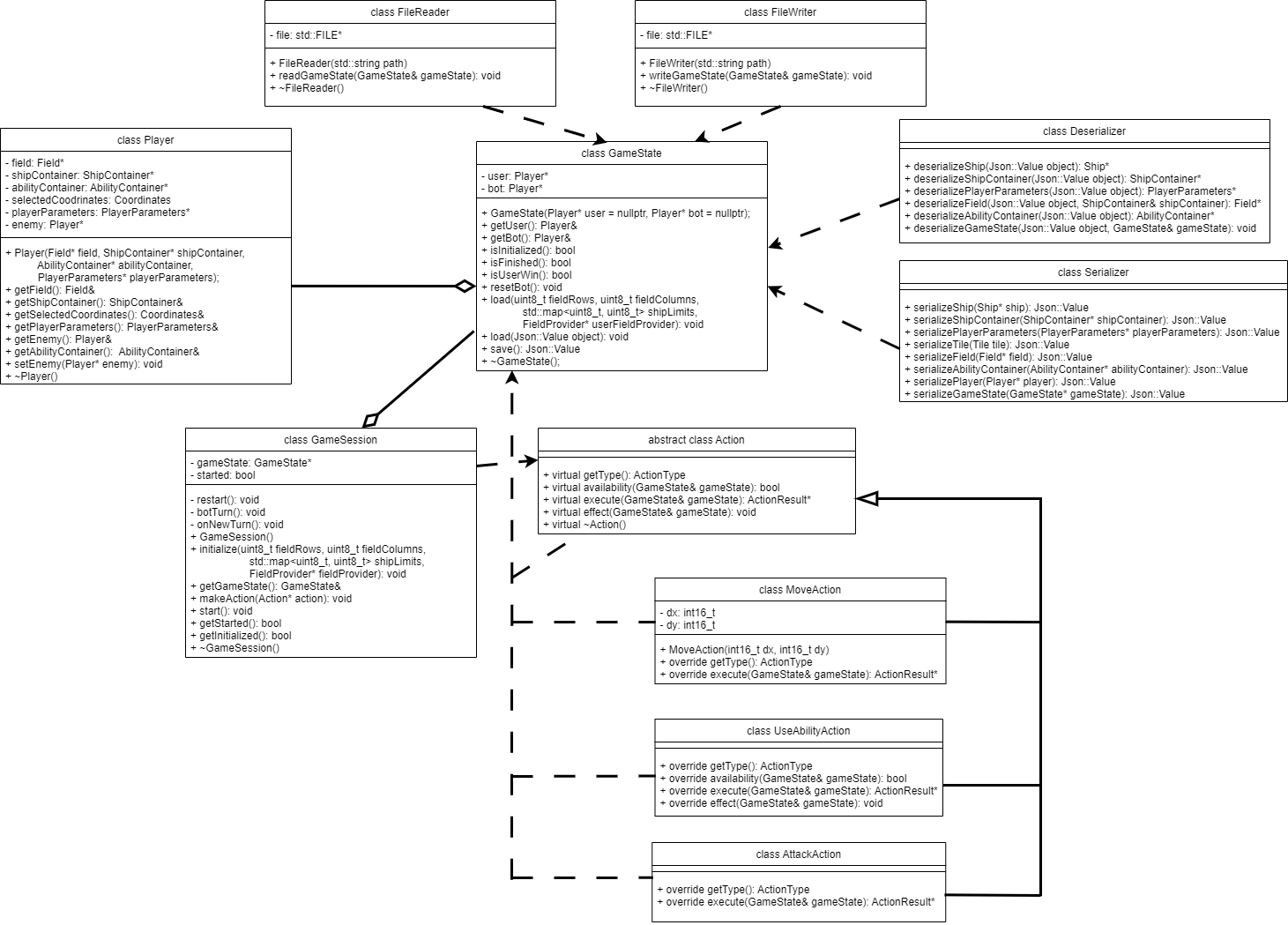


Рисунок 3 – UML-диаграмма реализованных классов

Приложение B

**Файл Action.hpp:**

#ifndef ACTION\_HPP // ACTION\_HPP

#define ACTION\_HPP

#include "ActionResult.hpp"

#include "GameState.hpp"

enum class ActionType { ONCE, MANY, TERMINATING };

class Action {

public:

virtual ActionType getType() = 0;

virtual bool availability(GameState& gameState) { return true; }

virtual ActionResult\* execute(GameState& gameState) = 0;

virtual void effect(GameState& gameState) {}

virtual ~Action() {}

};

#endif // ACTION\_HPP

**Файл ActionResult.hpp:**

#ifndef ACTIONRESULT\_HPP // ACTIONRESULT\_HPP

#define ACTIONRESULT\_HPP

#include <stdexcept>

#include <string>

class ActionResult {

public:

virtual std::string getResult() = 0;

virtual ~ActionResult() {}

};

class ActionResultSuccess : public ActionResult {

private:

std::string result;

public:

ActionResultSuccess(std::string result) : result(result) {}

std::string getResult() override;

};

class ActionResultFailure : public ActionResult {

private:

std::string result;

public:

ActionResultFailure(std::runtime\_error error) : result(error.what()) {}

ActionResultFailure(std::exception exception) : result(exception.what()) {}

std::string getResult() override;

};

#endif // ACTIONRESULT\_HPP

**Файл AttackAction.hpp:**

#ifndef ATTACKACTION\_HPP // ATTACKACTION\_HPP

#define ATTACKACTION\_HPP

#include "Action.hpp"

#include "Coordinates.hpp"

class AttackAction : public Action {

public:

ActionType getType() override;

ActionResult\* execute(GameState& gameState) override;

};

#endif // ATTACKACTION\_HPP

**Файл Deserializer.hpp:**

#ifndef DESERIALIZER\_HPP // DESERIALIZER\_HPP

#define DESERIALIZER\_HPP

#include <jsoncpp/json/json.h>

#include "AbilityContainer.hpp"

#include "Field.hpp"

#include "GameState.hpp"

#include "PlayerParameters.hpp"

#include "Ship.hpp"

#include "ShipContainer.hpp"

class Ship;

class ShipContainer;

class PlayerParameters;

class Field;

class AbilityContainer;

class Deserializer {

public:

Ship\* deserializeShip(Json::Value object);

ShipContainer\* deserializeShipContainer(Json::Value object);

PlayerParameters\* deserializePlayerParameters(Json::Value object);

Field\* deserializeField(Json::Value object, ShipContainer& shipContainer);

AbilityContainer\* deserializeAbilityContainer(Json::Value object);

void deserializeGameState(Json::Value object, GameState& gameState);

};

#endif // DESERIALIZER\_HPP

**Файл FileReader.hpp:**

#ifndef FILEREADER\_HPP // FILEREADER\_HPP

#define FILEREADER\_HPP

#include <fstream>

#include <jsoncpp/json/json.h>

#include <string>

#include "GameState.hpp"

class FileReader {

private:

std::FILE\* file;

public:

FileReader(std::string path);

void readGameState(GameState& gameState);

~FileReader();

};

#endif // FILEREADER\_HPP

**Файл FileWriter.hpp:**

#ifndef FILEWRITER\_HPP // FILEWRITER\_HPP

#define FILEWRITER\_HPP

#include <fstream>

#include <jsoncpp/json/json.h>

#include <string>

#include "GameState.hpp"

class FileWriter {

private:

std::FILE\* file;

public:

FileWriter(std::string path);

void writeGameState(GameState& gameState);

~FileWriter();

};

#endif // FILEWRITER\_HPP

**Файл GameSession.hpp:**

#ifndef GAMESESSION\_HPP // GAMESESSION\_HPP

#define GAMESESSION\_HPP

#include <vector>

#include "Action.hpp"

#include "FieldProvider.hpp"

#include "GameObserver.hpp"

#include "GameState.hpp"

#include "Player.hpp"

class GameSession {

private:

std::vector<GameObserver\*> observers;

GameState\* gameState;

bool started;

void restart();

void botTurn();

void onNewTurn();

public:

GameSession();

void initialize(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipLimits,

FieldProvider\* fieldProvider);

GameState& getGameState();

void makeAction(Action\* action);

void start();

bool getStarted();

bool getInitialized();

void addObserver(GameObserver\* observer);

~GameSession();

};

#endif // GAMESESSION\_HPP

**Файл GameState.hpp:**

#ifndef GAMESTATE\_HPP // GAMESTATE\_HPP

#define GAMESTATE\_HPP

#include "Deserializer.hpp"

#include "FieldProvider.hpp"

#include "Player.hpp"

#include "Serializer.hpp"

class Player;

class FieldProvider;

class GameState {

private:

Player\* user;

Player\* bot;

friend class Serializer;

friend class Deserializer;

public:

GameState(Player\* user = nullptr, Player\* bot = nullptr);

Player& getUser();

Player& getBot();

bool isInitialized();

bool isFinished();

bool isUserWin();

void resetBot();

void load(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipLimits,

FieldProvider\* userFieldProvider);

void load(Json::Value object);

Json::Value save();

~GameState();

};

std::ostream& operator<<(std::ostream& os, GameState& gameState);

std::istream& operator>>(std::istream& is, GameState& gameState);

#endif // GAMESTATE\_HPP

**Файл MoveAction.hpp:**

#ifndef SELECTACTION\_HPP // SELECTACTION\_HPP

#define SELECTACTION\_HPP

#include "Action.hpp"

class MoveAction : public Action {

private:

int16\_t dx;

int16\_t dy;

public:

MoveAction(int16\_t dx, int16\_t dy) : dx(dx), dy(dy) {}

ActionType getType() override;

ActionResult\* execute(GameState& gameState) override;

};

#endif // SELECTACTION\_HPP

**Файл Player.hpp:**

#ifndef PLAYER\_HPP // PLAYER\_HPP

#define PLAYER\_HPP

#include <jsoncpp/json/json.h>

#include "AbilityContainer.hpp"

#include "Field.hpp"

#include "PlayerParameters.hpp"

#include "Serializer.hpp"

#include "ShipContainer.hpp"

class AbilityContainer;

class Player {

private:

Field\* field;

ShipContainer\* shipContainer;

AbilityContainer\* abilityContainer;

Coordinates selectedCoodrinates;

PlayerParameters\* playerParameters;

Player\* enemy;

friend class Serializer;

public:

Player(Field\* field, ShipContainer\* shipContainer,

AbilityContainer\* abilityContainer,

PlayerParameters\* playerParameters);

Field& getField();

ShipContainer& getShipContainer();

Coordinates& getSelectedCoordinates();

PlayerParameters& getPlayerParameters();

Player& getEnemy();

AbilityContainer& getAbilityContainer();

void setEnemy(Player\* enemy);

~Player();

};

#endif // PLAYER\_HPP

**Файл Serializer.hpp:**

#ifndef SERIALIZER\_HPP // SERIALIZER\_HPP

#define SERIALIZER\_HPP

#include <jsoncpp/json/json.h>

#include "AbilityContainer.hpp"

#include "Field.hpp"

#include "GameState.hpp"

#include "PlayerParameters.hpp"

#include "Ship.hpp"

#include "ShipContainer.hpp"

#include "Tile.hpp"

#define CURRENT\_SCHEMA\_VERSION 1

class Ship;

class ShipContainer;

class PlayerParameters;

class Field;

class AbilityContainer;

class Player;

class GameState;

class Serializer {

public:

Json::Value serializeShip(Ship\* ship);

Json::Value serializeShipContainer(ShipContainer\* shipContainer);

Json::Value serializePlayerParameters(PlayerParameters\* playerParameters);

Json::Value serializeTile(Tile tile);

Json::Value serializeField(Field\* field);

Json::Value serializeAbilityContainer(AbilityContainer\* abilityContainer);

Json::Value serializePlayer(Player\* player);

Json::Value serializeGameState(GameState\* gameState);

};

#endif // SERIALIZER\_HPP

**Файл UseAbilityAction.hpp:**

#ifndef USEABILITYACTION\_HPP // USEABILITYACTION\_HPP

#define USEABILITYACTION\_HPP

#include "Action.hpp"

class UseAbilityAction : public Action {

public:

ActionType getType() override;

bool availability(GameState& gameState) override;

ActionResult\* execute(GameState& gameState) override;

void effect(GameState& gameState) override;

};

#endif // USEABILITYACTION\_HPP

**Файл AbilityResult.cpp:**

#include "../include/ActionResult.hpp"

std::string ActionResultSuccess::getResult() {

return result;

}

std::string ActionResultFailure::getResult() {

return result;

}

**Файл AttackAction.cpp:**

#include "../include/AttackAction.hpp"

ActionType AttackAction::getType() {

return ActionType::TERMINATING;

}

ActionResult\* AttackAction::execute(GameState& gameState) {

Player& user = gameState.getUser();

PlayerParameters& userPlayerParameters = user.getPlayerParameters();

uint8\_t damage = userPlayerParameters.getDamage();

Coordinates coordinates = user.getSelectedCoordinates();

try {

user.getEnemy().getField().shootAt(coordinates, damage);

} catch (std::runtime\_error e) {

return new ActionResultFailure(e);

}

userPlayerParameters.resetAttackParameters();

return new ActionResultSuccess("Attack was made.");

}

**Файл Deserializer.cpp:**

#include "../include/Deserializer.hpp"

Ship\* Deserializer::deserializeShip(Json::Value object) {

uint8\_t length = object["segments"].size();

Ship\* ship = new Ship(length);

ship->id = object["id"].asInt64();

ship->segments.clear();

for (uint8\_t i = 0; i < object["segments"].size(); ++i) {

ship->segments.push\_back(object["segments"][i].asUInt());

}

return ship;

}

ShipContainer\* Deserializer::deserializeShipContainer(Json::Value object) {

std::map<uint8\_t, uint8\_t> blankLimits = {};

ShipContainer\* shipContainer = new ShipContainer(blankLimits);

Json::Value shipsArray = object["ships"];

std::map<uint8\_t, uint8\_t> limits = {};

for (size\_t i = 0; i < shipsArray.size(); ++i) {

Ship\* ship = deserializeShip(shipsArray[(int)i]);

shipContainer->ships.push\_back(ship);

uint8\_t length = ship->getLength();

if (limits.count(length) == 0) {

limits[length] = 1;

} else {

limits[length] += 1;

}

}

shipContainer->limits = limits;

return shipContainer;

}

PlayerParameters\* Deserializer::deserializePlayerParameters(

Json::Value object) {

PlayerParameters\* playerParameters = new PlayerParameters();

playerParameters->damage = object["damage"].asUInt();

playerParameters->isAbilityUsed = object["isAbilityUsed"].asBool();

return playerParameters;

}

Field\* Deserializer::deserializeField(Json::Value object,

ShipContainer& shipContainer) {

uint8\_t rows = object["rows"].asUInt();

uint8\_t columns = object["columns"].asUInt();

Field\* field = new Field(rows, columns);

field->tiles.clear();

Json::Value tilesArray = object["tiles"];

for (size\_t i = 0; i < rows; ++i) {

field->tiles.push\_back(std::vector<Tile>());

for (size\_t j = 0; j < columns; ++j) {

int index = i \* rows + j;

TileState tileState =

static\_cast<TileState>(tilesArray[index]["tileState"].asInt());

Ship\* ship = nullptr;

if (!tilesArray[index]["shipId"].isNull()) {

ship = &shipContainer.getShipById(

tilesArray[index]["shipId"].asInt64());

}

field->tiles[i].push\_back(Tile(ship, tileState));

}

}

return field;

}

AbilityContainer\* Deserializer::deserializeAbilityContainer(

Json::Value object) {

AbilityContainer\* abilityContainer = new AbilityContainer();

uint8\_t abilitiesSize = abilityContainer->abilities.size();

for (uint8\_t i = 0; i < abilitiesSize; ++i) {

Ability\* ability = abilityContainer->abilities.front();

abilityContainer->abilities.pop\_front();

delete ability;

}

Json::Value abilitiesArray = object["abilities"];

for (int i = 0; i < abilitiesArray.size(); ++i) {

abilityContainer->abilities.push\_back(

abilityContainer->getAbilityFromString(

abilitiesArray[i].asString()));

}

return abilityContainer;

}

void Deserializer::deserializeGameState(Json::Value object,

GameState& gameState) {

if (!object.isMember("version")) {

throw std::runtime\_error("Corrupted savefile: no version found.");

}

if (object["version"].asInt64() != CURRENT\_SCHEMA\_VERSION) {

throw std::runtime\_error("Corrupted savefile: version mismatch.");

}

if (!object.isMember("hash")) {

throw std::runtime\_error("Corrupted savefile: no hash found.");

}

if (!object.isMember("game")) {

throw std::runtime\_error("Corrupted savefile: no game data found.");

}

Json::Value game = object["game"];

size\_t checkHash = std::hash<std::string>()(game.toStyledString());

if (object["hash"].asUInt64() != checkHash) {

throw std::runtime\_error("Corrupted savefile: hash mismatch.");

}

Json::Value players = game["players"];

Json::Value userObject = players[0];

Json::Value botObject = players[1];

ShipContainer\* userShipContainer =

deserializeShipContainer(userObject["shipContainer"]);

Field\* userField =

deserializeField(userObject["field"], \*userShipContainer);

PlayerParameters\* userPlayerParameters =

deserializePlayerParameters(userObject["playerParameters"]);

AbilityContainer\* userAbilityContainer =

deserializeAbilityContainer(userObject["abilityContainer"]);

ShipContainer\* botShipContainer =

deserializeShipContainer(botObject["shipContainer"]);

Field\* botField = deserializeField(botObject["field"], \*botShipContainer);

PlayerParameters\* botPlayerParameters =

deserializePlayerParameters(botObject["playerParameters"]);

if (gameState.user != nullptr) {

delete gameState.user;

}

if (gameState.bot != nullptr) {

delete gameState.bot;

}

gameState.user = new Player(userField, userShipContainer,

userAbilityContainer, userPlayerParameters);

gameState.bot =

new Player(botField, botShipContainer, nullptr, botPlayerParameters);

gameState.user->setEnemy(gameState.bot);

gameState.bot->setEnemy(gameState.user);

}

**Файл FileReader.cpp:**

#include "../include/FileReader.hpp"

FileReader::FileReader(std::string path) : file(std::fopen(path.c\_str(), "r")) {

if (file == nullptr) {

throw std::runtime\_error("File not found.");

}

}

void FileReader::readGameState(GameState& gameState) {

std::fseek(file, 0, SEEK\_END);

size\_t size = std::ftell(file);

char\* buffer = new char[size];

std::rewind(file);

std::fread(buffer, sizeof(char), size, file);

std::stringstream(buffer) >> gameState;

delete[] buffer;

}

FileReader::~FileReader() {

std::fclose(file);

}

**Файл FileWriter.cpp:**

#include "../include/FileWriter.hpp"

FileWriter::FileWriter(std::string path) : file(std::fopen(path.c\_str(), "w")) {

if (file == nullptr) {

throw std::runtime\_error("Could not open file.");

}

}

void FileWriter::writeGameState(GameState& gameState) {

std::stringstream ss;

ss << gameState;

if (std::fputs(ss.str().c\_str(), file) == EOF) {

throw std::runtime\_error("Could not write to file.");

}

}

FileWriter::~FileWriter() {

std::fclose(file);

}

**Файл GameSession.cpp:**

#include "../include/GameSession.hpp"

#include "../include/RandomFieldProvider.hpp"

GameSession::GameSession() {

gameState = new GameState();

started = false;

observers = std::vector<GameObserver\*>();

}

GameSession::~GameSession() {

if (gameState != nullptr) {

delete gameState;

}

}

void GameSession::start() {

if (!getInitialized()) {

throw std::runtime\_error("GameSession is not initialized.");

}

started = true;

for (auto observer : observers) {

observer->onUserTurn(\*gameState);

}

}

void GameSession::initialize(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipLimits,

FieldProvider\* userFieldProvider) {

if (getStarted()) {

throw std::runtime\_error("Trying to initialize started GameSession.");

}

gameState->load(fieldRows, fieldColumns, shipLimits, userFieldProvider);

}

GameState& GameSession::getGameState() {

return \*gameState;

}

bool GameSession::getStarted() {

return started;

}

bool GameSession::getInitialized() {

return gameState->isInitialized();

}

void GameSession::restart() {

gameState->resetBot();

}

void GameSession::botTurn() {

Player& bot = gameState->getBot();

Field& enemyField = bot.getEnemy().getField();

RandomGenerator random = RandomGenerator();

int16\_t x = random.randomBetween(0, enemyField.getColumns() - 1);

int16\_t y = random.randomBetween(0, enemyField.getRows() - 1);

Coordinates attackCoordinates = Coordinates(x, y);

uint8\_t damage = bot.getPlayerParameters().getDamage();

enemyField.shootAt(attackCoordinates, damage);

}

void GameSession::makeAction(Action\* action) {

if (!getStarted()) {

return;

}

if (!action->availability(\*gameState)) {

for (auto observer : observers) {

observer->onActionDenied();

}

delete action;

return;

}

ActionResult\* result = action->execute(\*gameState);

for (auto observer : observers) {

observer->onActionResult(result);

}

bool success = dynamic\_cast<ActionResultFailure\*>(result) == nullptr;

if (success) {

action->effect(\*gameState);

}

if (success && action->getType() == ActionType::TERMINATING) {

// проверка результатов хода и ход бота

if (gameState->isFinished() && gameState->isUserWin()) {

for (auto observer : observers) {

observer->onUserWin();

}

restart();

}

for (auto observer : observers) {

observer->onBotTurn();

}

botTurn(); // если нужно менять режим игры, см. сюда

if (gameState->isFinished() && !gameState->isUserWin()) {

// игра проиграна

for (auto observer : observers) {

observer->onBotWin();

}

started = false;

delete gameState;

gameState = new GameState();

} else {

onNewTurn();

}

}

delete result;

delete action;

}

void GameSession::onNewTurn() {

gameState->getUser().getPlayerParameters().resetAll();

for (auto observer : observers) {

observer->onUserTurn(\*gameState);

}

}

void GameSession::addObserver(GameObserver\* observer) {

if (observer == nullptr) {

return;

}

observers.push\_back(observer);

}

**Файл GameState.cpp:**

#include "../include/GameState.hpp"

#include <iostream>

#include "../include/RandomFieldProvider.hpp"

#include "../include/Serializer.hpp"

GameState::GameState(Player\* user, Player\* bot) : user(user), bot(bot) {}

Player& GameState::getUser() {

if (user == nullptr) {

throw std::runtime\_error(

"Trying to access user from (probably) not initialized GameState.");

}

return \*user;

}

Player& GameState::getBot() {

if (bot == nullptr) {

throw std::runtime\_error(

"Trying to access bot from (probably) not initialized GameState.");

}

return \*bot;

}

bool GameState::isInitialized() {

return user != nullptr && bot != nullptr;

}

bool GameState::isFinished() {

if (!isInitialized()) {

throw std::runtime\_error(

"Trying to access properties of (probably) not initialized "

"GameState.");

}

return user->getShipContainer().getAliveShipsCount() == 0 ||

bot->getShipContainer().getAliveShipsCount() == 0;

}

bool GameState::isUserWin() {

if (!isInitialized()) {

throw std::runtime\_error(

"Trying to access properties of (probably) not initialized "

"GameState.");

}

return bot->getShipContainer().getAliveShipsCount() == 0;

}

void GameState::resetBot() {

if (user == nullptr) {

throw std::runtime\_error(

"Cannot reset bot player due to null user player.");

}

if (bot != nullptr) {

delete bot;

}

uint8\_t fieldRows = user->getField().getRows();

uint8\_t fieldColumns = user->getField().getColumns();

std::map<uint8\_t, uint8\_t> shipLimits =

user->getShipContainer().getLimits();

Field\* botField = new Field(fieldRows, fieldColumns);

ShipContainer\* botShipContainer = new ShipContainer(shipLimits);

PlayerParameters\* botPlayerParameters = new PlayerParameters();

bot = new Player(botField, botShipContainer, nullptr, botPlayerParameters);

FieldProvider\* botFieldProvider = new RandomFieldProvider();

botFieldProvider->placeShips(\*botField, \*botShipContainer);

user->setEnemy(bot);

bot->setEnemy(user);

delete botFieldProvider;

}

GameState::~GameState() {

if (user != nullptr) {

delete user;

}

if (bot != nullptr) {

delete bot;

}

}

void GameState::load(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipLimits,

FieldProvider\* userFieldProvider) {

if (userFieldProvider == nullptr) {

throw std::runtime\_error("No FieldProvider provided.");

}

if (user != nullptr) {

delete user;

}

Field\* userField = new Field(fieldRows, fieldColumns);

ShipContainer\* userShipContainer = new ShipContainer(shipLimits);

AbilityContainer\* userAbilityContainer = new AbilityContainer();

PlayerParameters\* userPlayerParameters = new PlayerParameters();

userFieldProvider->placeShips(\*userField, \*userShipContainer);

user = new Player(userField, userShipContainer, userAbilityContainer,

userPlayerParameters);

resetBot();

}

void GameState::load(Json::Value object) {

try {

Deserializer().deserializeGameState(object, \*this);

} catch (Json::LogicError e) {

throw std::runtime\_error("Corrupted savefile: malformed json.");

}

}

Json::Value GameState::save() {

return Serializer().serializeGameState(this);

}

std::ostream& operator<<(std::ostream& os, GameState& gameState) {

os << gameState.save();

return os;

}

std::istream& operator>>(std::istream& is, GameState& gameState) {

Json::Value object;

try {

is >> object;

} catch (Json::RuntimeError e) {

throw std::runtime\_error("Corrupted savefile: malformed json.");

}

gameState.load(object);

return is;

}

**Файл MoveAction.cpp:**

#include "../include/MoveAction.hpp"

ActionType MoveAction::getType() {

return ActionType::MANY;

}

ActionResult\* MoveAction::execute(GameState& gameState) {

Player& user = gameState.getUser();

Coordinates& coordinates = user.getSelectedCoordinates();

coordinates.x += dx;

coordinates.y += dy;

return new ActionResultSuccess(

"Moved to x: " + std::to\_string(coordinates.x) +

", y: " + std::to\_string(coordinates.y));

}

**Файл Player.cpp:**

#include "../include/Player.hpp"

Player::Player(Field\* field, ShipContainer\* shipContainer,

AbilityContainer\* abilityContainer,

PlayerParameters\* playerParameters)

: field(field),

shipContainer(shipContainer),

playerParameters(playerParameters),

abilityContainer(abilityContainer) {

selectedCoodrinates = Coordinates();

}

Player::~Player() {

if (field != nullptr) {

delete field;

}

if (shipContainer != nullptr) {

delete shipContainer;

}

if (abilityContainer != nullptr) {

delete abilityContainer;

}

if (playerParameters != nullptr) {

delete playerParameters;

}

}

Field& Player::getField() {

if (field == nullptr) {

throw std::runtime\_error("Trying to access null Field.");

}

return \*field;

}

ShipContainer& Player::getShipContainer() {

if (shipContainer == nullptr) {

throw std::runtime\_error("Trying to access null ShipContainer.");

}

return \*shipContainer;

}

Coordinates& Player::getSelectedCoordinates() {

return selectedCoodrinates;

}

PlayerParameters& Player::getPlayerParameters() {

if (playerParameters == nullptr) {

throw std::runtime\_error("Trying to access null PlayerParameters.");

}

return \*playerParameters;

}

Player& Player::getEnemy() {

if (enemy == nullptr) {

throw std::runtime\_error("Trying to access null Player(enemy).");

}

return \*enemy;

}

AbilityContainer& Player::getAbilityContainer() {

if (abilityContainer == nullptr) {

throw std::runtime\_error("Trying to access null AbilityContainer.");

}

return \*abilityContainer;

}

void Player::setEnemy(Player\* enemy) {

if (enemy == nullptr) {

throw std::runtime\_error("Trying to set null Player(enemy).");

}

this->enemy = enemy;

if (abilityContainer != nullptr) {

enemy->getField().addObserver(abilityContainer);

}

}

**Файл Serializer.cpp:**

#include "../include/Serializer.hpp"

Json::Value Serializer::serializeShip(Ship\* ship) {

Json::Value object = Json::Value();

object["id"] = ship->id;

for (uint8\_t i = 0; i < ship->segments.size(); ++i) {

object["segments"][i] = ship->segments[i];

}

return object;

}

Json::Value Serializer::serializeShipContainer(ShipContainer\* shipContainer) {

Json::Value object = Json::Value();

for (int i = 0; i < shipContainer->ships.size(); ++i) {

object["ships"][i] = serializeShip(shipContainer->ships[i]);

}

return object;

}

Json::Value Serializer::serializePlayerParameters(

PlayerParameters\* playerParameters) {

Json::Value object = Json::Value();

object["damage"] = playerParameters->damage;

object["isAbilityUsed"] = playerParameters->isAbilityUsed;

return object;

}

Json::Value Serializer::serializeTile(Tile tile) {

Json::Value object = Json::Value();

object["tileState"] = static\_cast<int>(tile.tileState);

if (tile.ship == nullptr) {

object["shipId"] = Json::nullValue;

} else {

object["shipId"] = tile.ship->getId();

}

return object;

}

Json::Value Serializer::serializeField(Field\* field) {

Json::Value object = Json::Value();

object["rows"] = field->rows;

object["columns"] = field->columns;

for (size\_t i = 0; i < field->rows; ++i) {

for (size\_t j = 0; j < field->columns; ++j) {

int index = i \* field->rows + j;

object["tiles"][index] = serializeTile(field->tiles[i][j]);

}

}

return object;

}

Json::Value Serializer::serializeAbilityContainer(

AbilityContainer\* abilityContainer) {

Json::Value object = Json::Value();

object["abilities"] = Json::arrayValue;

for (int i = 0; i < abilityContainer->abilities.size(); ++i) {

object["abilities"][i] = abilityContainer->abilities[i]->getName();

}

return object;

}

Json::Value Serializer::serializePlayer(Player\* player) {

Json::Value object = Json::Value();

if (player->playerParameters != nullptr) {

object["playerParameters"] =

serializePlayerParameters(player->playerParameters);

}

if (player->shipContainer != nullptr) {

object["shipContainer"] = serializeShipContainer(player->shipContainer);

}

if (player->field != nullptr) {

object["field"] = serializeField(player->field);

}

if (player->abilityContainer != nullptr) {

object["abilityContainer"] =

serializeAbilityContainer(player->abilityContainer);

}

return object;

}

Json::Value Serializer::serializeGameState(GameState\* gameState) {

Json::Value game = Json::Value();

game["players"][0] = serializePlayer(gameState->user);

game["players"][1] = serializePlayer(gameState->bot);

size\_t hash = std::hash<std::string>()(game.toStyledString());

Json::Value object = Json::Value();

object["hash"] = hash;

object["version"] = CURRENT\_SCHEMA\_VERSION;

object["game"] = game;

return object;

}

**Файл UseAbilityAction.cpp:**

#include "../include/UseAbilityAction.hpp"

ActionType UseAbilityAction::getType() {

return ActionType::ONCE;

}

bool UseAbilityAction::availability(GameState& gameState) {

return !gameState.getUser().getPlayerParameters().getIsAbilityUsed();

}

ActionResult\* UseAbilityAction::execute(GameState& gameState) {

AbilityResult\* abilityResult;

try {

Player& user = gameState.getUser();

abilityResult = user.getAbilityContainer().useAbility(user);

} catch (std::runtime\_error e) {

return new ActionResultFailure(e);

}

ActionResult\* result = new ActionResultSuccess(abilityResult->getResult());

delete abilityResult;

return result;

}

void UseAbilityAction::effect(GameState& gameState) {

gameState.getUser().getPlayerParameters().setIsAbilityUsed(true);

}