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. 3383 _____ Солдунова Е.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы

Изучение связывания классов в объектно-ориентированном программировании для реализации модифицированной игры “Морской бой” в соответствии с её принципами на языке C++.

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

1. Подключение необходимых библиотек

- *#include <fstream>* – библиотека для чтения и записи данных в файл.

2. Описание реализованных классов и их методов

a. Класс *Game:*

Класс Game содержит в полях основные атрибуты игры, такие как поля игрока и противника, корабли игрока и противника, неатакованные клетки и способности игрока.

Методы класса:

- Конструктор *Game()*:

Инициализирует поля игрока и противника размером 2x2 и пустыми кораблями. Проверяет, нужно ли загружать игру из сохранения или начать новую: если игра загружается из сохранения, то загружает состояние игры и выводит сообщение об успешной загрузке.

Метод запрашивает у пользователя размеры поля игрока, количество кораблей и их размеры, а затем размещает корабли на поле. Инициализирует способности игрока.

- Метод *StartGame()*:

Начинает новую игру. В цикле выполняет раунды игры, пока противник не победит. В случае победы противника запускает новую игру.

- Метод *Round()*:

Инициализирует новое поле и корабли противника, в размере и количестве поля и кораблей игрока. Размещает корабли на поле противника случайным образом. Инициализирует массив неатакованных клеток. Выполняет ход игрока и противника поочередно, пока один из них не победит. Возвращает результат раунда (победа игрока, поражение игрока или игра не завершена).

- Метод *PlayerMove()*:

Предлагает игроку использовать способность: если игрок соглашается, применяет способность к полю противника.

Производит атаку клетки поля противника, введённую игроком. После каждого хода предлагает сохранить игру.

- Метод *EnemyMove():*

Выбирает случайные координаты для атаки на поле игрока. Проверяет, не была ли выбранная клетка атакована ранее. Выполняет атаку на поле игрока и обрабатывает результат.

- Метод *InputInts(int num)*:

Запрашивает ввод целочисленных значений от пользователя и возвращает введенные значения в виде вектора. Используется для избегания дублирования кода при вводе различных данных пользователем. Метод используется для получения координат клеток, размеров кораблей в других методах.

b. Класс *GameState:*

Класс *GameState* представляет игровое состояние, включающее в себя информацию о расстановке кораблей, полях, способностях.

- Конструктор класса *GameState:*

Принимает объект типа Game и инициализирует менеджеры кораблей для игрока и врага, игровые поля для игрока и врага, а также менеджер способностей и список неатакованных клеток.

- Метод *save(const string& filename):*

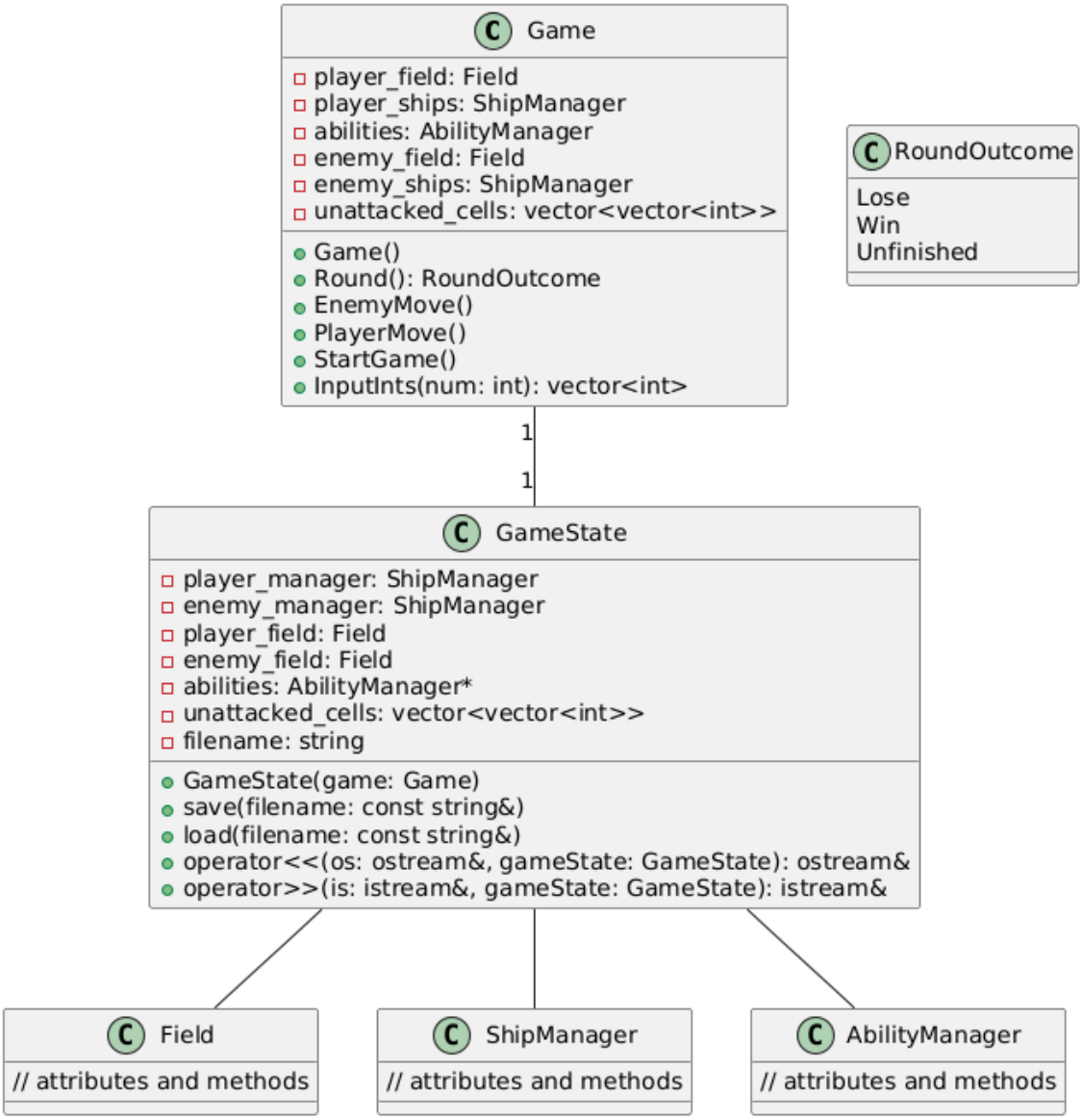
Сохраняет текущее состояние игры в файл с указанным именем. Метод открывает файл для записи, записывает информацию о состоянии игры в файл и закрывает файл.

- Метод l*oad(const string& filename):*

Загружает игровое состояние из файла с указанным именем. Метод открывает файл для чтения, считывает информацию о состоянии игры из файла и закрывает файл.

Операторы << и >> перегружены для сериализации и десериализации объекта *GameState*. Оператор << записывает информацию о состоянии игры в поток вывода, включая размеры полей, расстановку кораблей, статус клеток на поле и список неатакованных клеток. Оператор >> считывает информацию из потока ввода и создает объект *GameState* на основе этой информации.

UML реализованных классов



Вывод

В ходе выполнения лабораторной работы было изучено и реализовано связывание классов для создания модифицированной игры “Морской бой” на языке C++.

ПРИЛОЖЕНИЕ А

Исходный код программы

Game.h

```
#ifndef GAME_H
#define GAME_H

#include "Ship.h"
#include "ShipManager.h"
#include "Field.h"
#include "Ability.h"
#include "AbilityManager.h"
#include "Exceptions.h"
#include "GameState.h"

enum class RoundOutcome {Lose, Win, Unfinished};

class Game{
public:
Field player_field;
ShipManager player_ships;
AbilityManager abilities;
Field enemy_field;
ShipManager enemy_ships;
vector<vector<int>> unattacked_cells;
Game();
RoundOutcome Round();
void EnemyMove();
void PlayerMove();
void StartGame();
vector<int> InputInts(int num);
};
#endif
```

Game.cpp

```
#include "Game.h"

Game::Game():player_field(2,      2),      player_ships(0,      {}),
enemy_field(2, 2), enemy_ships(0, {}), unattacked_cells({}){
string loading;
cout << "Do you want to load the game? Yes - enter yes, No - enter
any line: " << endl;
cin >> loading;
if (loading == "yes"){
GameState game_state(this);
game_state.load("saving");
cout << "Your game is succesfully load!" << endl;
}

cout << "Hello! Before starting the new game, set the height and
width of the fields:" << endl;
vector<int> sizes = InputInts(2);
while (true){
try{
Field new_field(sizes[0], sizes[1]);
break;
}
catch (const FieldSizeException& ex){
cout << ex.getMessage() << endl;
sizes = InputInts(2);
}
}

player_field = Field(sizes[0], sizes[1], FieldType::Users);
cout << "The creation of the field was successful!" << endl;
```

```
cout << "Enter the number of ships:" << endl;
int quantity = InputInts(1)[0];
while((quantity<1)||(quantity>(player_field.GetWidth()*player_field
.GetHeight()/2))){
cout << "The quantity of the ships cannot be negative or zero,
please try again: ";
quantity = InputInts(1)[0];
}
player_field.Draw();

cout << "Enter sizes of ships. After each size, determine the
position of the ship: enter v if you want the ship to be positioned
vertically or h if horizontally" << endl;
vector<int> ships(quantity);
vector<char> orient(quantity);
int i = 0;
player_ships = ShipManager(0, {});

while (i < quantity){
ships[i] = InputInts(1)[0];
if (ships[i] < 1){cout << "The length of the ship cannot be
negative or zero, please try again: " << endl;}
else {
while(!(cin >> orient[i])){
                    cout << "The entered data is incorrect, please
try again: ";
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(),
'\n');
}
if (orient[i]=='v'){player_ships.AddShip(ships[i]);}
if                   (orient[i]=='h'){player_ships.AddShip(ships[i],
Orientation::Horizontal);}
```

```
if ((orient[i]!='v')&&(orient[i]!='h')){
cout << "You entered an incorrect orientation symbol, so the ship is set vertically by default." << endl;
player_ships.AddShip(ships[i]);
}
i++;
}
}

cout << "For each of the created ships, set the coordinates of its heads:" << endl;
i = 0;
int flag = 0;
vector<int> coordinates;
while (i < quantity){
coordinates = InputInts(2);
try{
player_field.PutShip(coordinates[0], coordinates[1], player_ships.GetShip(i));
i++;
}
catch (const IncorrectPlacementException& ex){
cout << ex.getMessage() << endl;
flag++;
if (flag == 3){
player_ships.DeleteShip(i);
cout << "You have placed the ship incorrectly three times. Perhaps it was too big, so it was deleted. You can continue to place the remaining ships." << endl;
flag = 0;
i++;
}
}
```

```
}
player_field.Draw();
abilities = AbilityManager();
}

void Game::StartGame(){
cout << "The new game has started!" << endl;
while (this->Round() != RoundOutcome::Lose){
this->Round();
}
//Game new_game;
//new_game.StartGame();
}

RoundOutcome Game::Round(){
Field new_field(player_field.GetHeight(), player_field.GetWidth(),
FieldType::Enemy);
enemy_field = new_field;
enemy_field.Draw();
vector<int> enemy_ships_size(player_ships.GetShipsNumber());
for (int i = 0; i < player_ships.GetShipsNumber();
i++){enemy_ships_size[i] = (*player_ships.GetShip(i)).GetLenght();}
ShipManager new_ships = ShipManager(player_ships.GetShipsNumber(),
enemy_ships_size);
enemy_ships = new_ships;
int x;
int y;
int i = 0;
int flag = 0;
while (i < enemy_ships.GetShipsNumber()){
x = rand() % enemy_field.GetHeight();
y = rand() % enemy_field.GetWidth();
```

```
try{
enemy_field.PutShip(x, y, enemy_ships.GetShip(i));
i++;
flag = 0;
}
catch (const IncorrectPlacementException& ex){
flag++;
if (flag == 3){
enemy_ships.DeleteShip(i);
flag = 0;
i++;
}
}
}
vector<vector<int>>                cells(player_field.GetHeight(),
vector<int>(player_field.GetWidth(), 1));
unattacked_cells = cells;
while (true){
PlayerMove();
enemy_field.Draw();
string saving;
cout << "Do you want to save the game? Yes - enter yes, No - enter
any line: " << endl;
cin >> saving;
if (saving == "yes"){
GameState game_state(this);
game_state.save("saving");
}
if (!enemy_ships.GetFleetState()) {return RoundOutcome::Win;}
EnemyMove();
player_field.Draw();
if (!player_ships.GetFleetState()) {return RoundOutcome::Lose;}
}
```

```
return RoundOutcome::Unfinished;
}

void Game::PlayerMove(){
string ability;
cout << "Do you want to apply the ability? Yes - enter yes, No -
enter any line: " << endl;
cin >> ability;
if (ability == "yes"){
try{
if   (abilities.ApplicationOfAbilities(enemy_field,   enemy_ships))
abilities.AddAbility();
}
catch  (const  NoAbilityException&  ex){cout  <<  ex.getMessage()  <<
endl;}
}
cout << "Enter the coordinates of the cell you want to attack:" <<
endl;
vector<int> coordinates = InputInts(2);
int x = coordinates[0];
int y = coordinates[1];
try{
if(enemy_field.Attack(x, y)){
if(enemy_field.FieldGetShipStatus(x,     y)==ShipStatus::Destroyed)
abilities.AddAbility();
}
}
catch (const OutsideAttackException& ex){cout << ex.getMessage() <<
endl;}
}

void Game::EnemyMove(){
```

```
int attack_y = rand() % player_field.GetHeight();
int attack_x = rand() % player_field.GetWidth();
while (unattacked_cells[attack_y][attack_x] != 1){
attack_y = rand() % player_field.GetHeight();
attack_x = rand() % player_field.GetWidth();
}
if (player_field.Attack(attack_x, attack_y)){
if    (player_field.FieldGetShipStatus(attack_x,    attack_y)    ==
ShipStatus::Destroyed){unattacked_cells[attack_y][attack_x] = 0;}
}
}

vector<int> Game::InputInts(int num){
vector<int> values(num);
if (num == 2){
while(!(cin >> values[0] >> values[1])){
cout << "The entered data is incorrect, please try again: ";
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
}
if (num == 1){
while(!(cin >> values[0])){
cout << "The entered data is incorrect, please try again: ";
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
}
return values;
}
```

GameState.h

```
#ifndef GAMESTATE_H
#define GAMESTATE_H

#include <fstream>
#include "Game.h"

class GameState{
public:
    ShipManager player_manager;
    ShipManager enemy_manager;
    Field player_field;
    Field enemy_field;
    AbilityManager* abilities;
    vector<vector <int>> unattacked_cells;
    string filename = "saving";

    GameState(Game game);
    void save(const string& filename);
    void load(const string& filename);
    friend ostream& operator<<(ostream& os, GameState& gameState);
    friend istream& operator>>(istream& is, GameState& gameState);
};

#endif
```

GameState.cpp

```
#include "GameState.h"

GameState::GameState(Game game): player_manager(game.player_ships),
enemy_manager(game.enemy_ships),   player_field(game.player_field),
enemy_field(game.enemy_field),          abilities(&game.abilities),
unattacked_cells(game.unattacked_cells){}
```

```
void GameState::save(const string& filename) {
ofstream outFile(filename);
        if (outFile.is_open()) {
            outFile << *this;
            outFile.close();
        }
        else {
            cerr << "File not open!" << endl;
        }
}

void GameState::load(const string& filename) {
        ifstream inFile(filename);
        if (inFile.is_open()) {
            inFile >> *this;
            inFile.close();
        } else {
            cerr << "File not open!" << endl;
        }
}

ostream& operator<<(ostream& os, GameState& gameState) {
os << "saving state" << "\n";
os    <<    gameState.player_field.GetHeight()    <<    "    "    <<
gameState.player_field.GetWidth() << "\n";
os << gameState.player_manager.GetShipsNumber() << "\n";
for  (int  i  =  0;  i  <  gameState.player_manager.GetShipsNumber();
i++){
os    <<    gameState.player_field.heads[i][0]    <<    "    "    <<
gameState.player_field.heads[i][1] << " ";
```

```
if    ((*gameState.player_manager.GetShip(i)).GetOrientation()    ==
Orientation::Vertical){os << "v ";}
else {os << "h ";}
os << (*gameState.player_manager.GetShip(i)).GetLenght() << " ";
for         (int         j         =         0;         j         <
(*gameState.player_manager.GetShip(i)).GetLenght();    i++){os    <<
(*gameState.player_manager.GetShip(i)).GetSegmentStatus(j) << " ";}
os << "\n";
}
os << (*gameState.abilities).GetLength() << "\n";
os << gameState.enemy_manager.GetShipsNumber() << "\n";
for (int i = 0; i < gameState.enemy_manager.GetShipsNumber(); i++){
os    <<    gameState.enemy_field.heads[i][0]    <<    "    "    <<
gameState.enemy_field.heads[i][1] << " ";
if    ((*gameState.enemy_manager.GetShip(i)).GetOrientation()    ==
Orientation::Vertical){os << "v ";}
else {os << "h ";}
for         (int         j         =         0;         j         <
(*gameState.enemy_manager.GetShip(i)).GetLenght();    i++){os    <<
(*gameState.enemy_manager.GetShip(i)).GetSegmentStatus(j) << " ";}
os << "\n";
}
for (int i = 0; i < gameState.enemy_field.GetHeight(); i++){
for (int j = 0; j < gameState.enemy_field.GetWidth(); j++){
if       (gameState.enemy_field.GetCellStatus(j,       i)       ==
Status::Unknown){os << "? ";}
else {os << "+ ";}
}
os << "\n";
}
for (int i = 0; i < gameState.unattacked_cells.size(); i++){
for (int j = 0; j < gameState.unattacked_cells[0].size(); j++){os
<< gameState.unattacked_cells[i][j] << " ";}
```

```
}
return os;
}

istream& operator>>(istream& is, GameState& gameState) {
        int height;
        int width;
is >> height >> width;
gameState.player_field = Field(height, width, FieldType::Users);
        gameState.enemy_field      =      Field(height,      width,
FieldType::Users);

        int player_ships_number;
is >> player_ships_number;

gameState.player_manager = ShipManager({0, {}});
char orientation;
int ship_len;
vector<int> head(2);

for (int i = 0; i < player_ships_number; i++){
is >> head[0] >> head[1];
is >> orientation;
is >> ship_len;
vector<int> ship_segments(ship_len);
for (int j = 0; j < ship_len; j++){is >> ship_segments[j];}
if (orientation == 'v'){gameState.player_manager.AddShip(ship_len,
Orientation::Vertical, ship_segments);}
else{gameState.player_manager.AddShip(ship_len,
Orientation::Horizontal, ship_segments);}
gameState.player_field.PutShip(head[0],                    head[1],
gameState.player_manager.GetShip(i));
```

```
}

int ability_quantity;
is >> ability_quantity;
AbilityManager manager;
gameState.abilities = &manager;
        for (size_t i = 0; i < ability_quantity; i++){
            (*gameState.abilities).AddAbility();
        }
int enemy_ships_number;
is >> enemy_ships_number;

gameState.enemy_manager = ShipManager({0, {}});

for (int i = 0; i < enemy_ships_number; i++){
is >> head[0] >> head[1];
is >> orientation;
is >> ship_len;
vector<int> ship_segments(ship_len);
for (int j = 0; j < ship_len; j++){is >> ship_segments[j];}
if  (orientation  ==  'v'){gameState.enemy_manager.AddShip(ship_len,
Orientation::Vertical, ship_segments);}
else{gameState.enemy_manager.AddShip(ship_len,
Orientation::Horizontal, ship_segments);}
gameState.enemy_field.PutShip(head[0],                    head[1],
gameState.enemy_manager.GetShip(i));
}

for (int i = 0; i < gameState.enemy_field.GetHeight(); i++){
for (int j = 0; j < gameState.enemy_field.GetWidth(); j++){
char status;
is >> status;
```

```
if (status == '?'){gameState.enemy_field.ChangeStatusUnknown(j,
i);}
}
}
gameState.enemy_field.ChangeFieldType();

vector<vector<int>>
new_unattacked_cells(gameState.enemy_field.GetHeight(),
vector<int>(gameState.enemy_field.GetWidth()));
for (int i = 0; i < gameState.enemy_field.GetHeight(); i++){
for (int j = 0; j < gameState.enemy_field.GetWidth(); j++){is >>
new_unattacked_cells[i][j];}
}
gameState.unattacked_cells = new_unattacked_cells;
return is;
}
```