**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Шаблонные классы

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Тукалкин.В.А |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создать шаблонные классы управления игрой и отображения игры, и классы для CLI.

## Задание.

a) Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.

b) Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.

c) Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.

d) Реализовать класс, отвечающий за отрисовку поля.

Примечание:

* Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
* После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
* Для представления команды можно разработать системы классов или использовать перечисление enum.
* Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
* При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

## Выполнение работы

Класс GameRenderer:

В приватном поле хранит указатель на игру и указатель на renderer.

1) GameRenderer(Game& \_game, Renderer& \_renderer) – конструктор класса.

2) void render() – выводит поле игрока и поле врага.

3) void renderPlayer() – выводит только поле игрока.

4) void print(const std::string& str) - выводит строку.

Класс GameLoop:

1) GameLoop() – конструктор

2) void startGame() – запускает цикл игры.

Класс GameController:

В приватном поле хранит указатель на игру и указатель на класс вывода.

1) GameController(Game& \_game, InputHandler& \_inputHandler) – конструктор класса.

2) Interaction processInput() – обработка команд ввода.

3) bool placeShip(int indexShip) – расстановка кораблей.

4) int check(int x) – проверка корректности координаты.

Класс TerminalInputHandler:

В приватном поле хранит карту команд и метод загрузки команд из файла.

1) void loadCommands(const std::string& filename) – загружает команды из файла, иначе стандартные.

2) TerminalInputHandler() – конструктор.

3) Command getKey() – получить команду.

4) tuple<int, int> getCoords() – получить координаты для атаки.

5) tuple<int, int, int> getCoordsShip() – получить параметры для корабля.

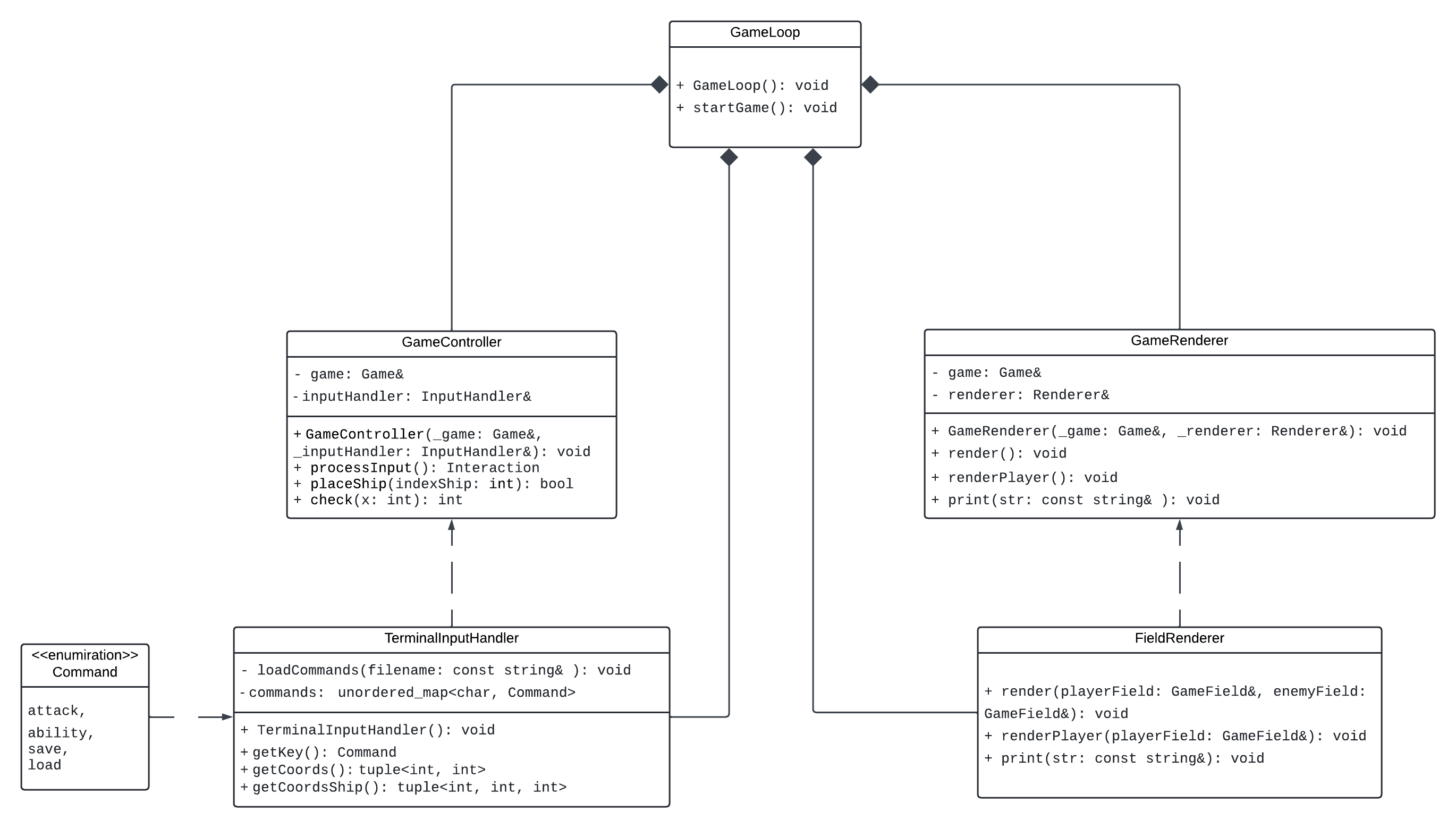
Класс FieldRenderer:

1) void render(GameField& playerField, GameField& enemyField) – выводит в консоль поля игрока и врага.

2) void renderPlayer(GameField& playerField) – выводит в консоль поле игрока.

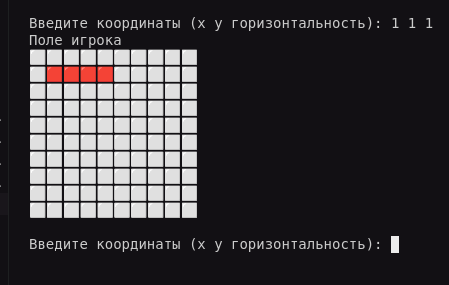
3) void print(const std::string& str) – выводит строку.

UML-диаграмма классов программы на данный момент:



## Тестирование.

Проверка расстановки кораблей:



Проверка интерфейса в цикле игры:



Проверка игры:



## Выводы

Были созданы шаблонные классы для игры и теперь можно сделать GUI для игры.

# Приложение А Исходный код программы

Название файла: main.cpp

#include "GameLoop.h"

int main() {

GameLoop game;

game.startGame();

return 0;

}

Название файла: FieldRenderer.h

#ifndef FIELDRENDERER\_H

#define FIELDRENDERER\_H

#include "GameField.h"

#include <string>

class FieldRenderer{

public:

void render(GameField& playerField, GameField& enemyField);

void renderPlayer(GameField& playerField);

void print(const std::string& str);

};

#endif // FIELDRENDERER\_H

Название файла: FieldRenderer.cpp

#include "FieldRenderer.h"

#include <iostream>

void FieldRenderer::render(GameField& playerField, GameField& enemyField){

std::cout << "\nПоле игрока\n";

playerField.printField(false);

std::cout << "\n\nПоле противника\n";

enemyField.printField(false);

}

void FieldRenderer::renderPlayer(GameField& playerField){

std::cout << "Поле игрока\n";

playerField.printField(false);

}

void FieldRenderer::print(const std::string& str){

std::cout << str;

}

Название файла: GameController.h

#ifndef GAMECONTROLLER\_H

#define GAMECONTROLLER\_H

#include "Game.h"

#include <functional>

#include <unordered\_map>

#include <string>

enum Command{

attack,

ability,

save,

load

};

template <typename InputHandler>

class GameController{

public:

GameController(Game& \_game, InputHandler& \_inputHandler) : game(\_game), inputHandler(\_inputHandler) {

/\*commands['S'] = std::bind

commands['A'] = std::bind

commands['s'] = std::bind(&Game::saveGame, &game, "save.json");

commands['l'] = std::bind(&Game::loadGame, &game, "save.json");\*/

}

Interaction processInput(){

Command key = inputHandler.getKey();

std::tuple<int, int> coords;

Interaction flag;

switch (key)

{

case Command::attack:

coords = inputHandler.getCoords();

std::get<0>(coords)=check(std::get<0>(coords));

std::get<1>(coords)=check(std::get<1>(coords));

flag=game.playerTurn(std::get<0>(coords), std::get<1>(coords), 0);

return flag;

break;

case Command::ability:

coords = inputHandler.getCoords();

std::get<0>(coords)=check(std::get<0>(coords));

std::get<1>(coords)=check(std::get<1>(coords));

flag=game.playerTurn(std::get<0>(coords), std::get<1>(coords), 1);

if(flag == -1) return Interaction::no\_ability;

break;

case Command::save:

game.saveGame("save.json");

return Interaction::save\_game;

break;

case Command::load:

game.loadGame("save.json");

return Interaction::load\_game;

break;

default:

break;

}

return Interaction::empty;

/\*

char key = inputHandler.getKey();

if (commands.find(key) != commands.end()) {

commands[key]();

}\*/

}

bool placeShip(int indexShip){

std::tuple<int, int, int> coords;

std::get<0>(coords)=check(std::get<0>(coords));

std::get<1>(coords)=check(std::get<1>(coords));

if(std::get<2>(coords)!=0) std::get<2>(coords)=1;

coords = inputHandler.getCoordsShip();

bool flag=game.placeShips(indexShip, std::get<0>(coords), std::get<1>(coords), std::get<2>(coords));

return flag;

}

int check(int x){

if(x>=0 && x<10) return x;

return 0;

}

private:

Game& game;

InputHandler& inputHandler;

//std::unordered\_map<char, std::function<void()>> commands;

};

#endif // GAMECONTROLLER\_H

Название файла: GameLoop.cpp

#include "GameLoop.h"

#include <cstdlib>

GameLoop::GameLoop() {}

void GameLoop::startGame(){

std::system("clear");

Game game;

TerminalInputHandler inputHandler;

FieldRenderer fieldRenderer;

GameController<TerminalInputHandler> gameController(game, inputHandler);

GameRenderer<FieldRenderer> gameRenderer(game, fieldRenderer);

gameRenderer.print("Введите число \n1 = Новая игра \n2 = Новая игра со случайной расстановкой \n3 = Загрузить сохранение\n");

int a=-1;

game.startGame();

while(1){

std::cin >> a;

if(a == 3){

game.loadGame("save.json");

break;

}else if(a == 2){

game.autoplaceShipsPlayer();

break;

}else if(a==1){

std::system("clear");

for(int i=0; i<10;i++){

bool flag=false;

while(!flag){

flag=gameController.placeShip(i);

if(!flag) gameRenderer.print("Некорректное расположение корабля\n");

gameRenderer.renderPlayer();

}

}

break;

}else{

gameRenderer.print("Введите корректный аргумент\n");

std::cin >> a;

}

}

std::system("clear");

Interaction command;

while (1) {

gameRenderer.print("\nКоличество способностей: ");

gameRenderer.print(std::to\_string(game.getAbilities()));

if(game.checkGameOver()) break;

gameRenderer.render();

command=gameController.processInput();

std::system("clear");

switch(command){

case Interaction::no\_ability:

gameRenderer.print("Способности закончились, будет заменено на атаку\n");

break;

case Interaction::miss:

gameRenderer.print("\n\033[3;42;30mПромах!\033[0m\n");

break;

case Interaction::shoot\_ship:

gameRenderer.print("\n\033[3;42;30mПопадание!\033[0m\n");

break;

case Interaction::destroy\_ship:

gameRenderer.print("\n\033[3;42;30mКорабль уничтожен!\033[0m\n");

break;

case Interaction::save\_game:

continue;

case Interaction::load\_game:

continue;

}

game.startRound();

}

}

Название файла: GameLoop.h

#include "GameController.h"

#include "GameRenderer.h"

#include "TerminalInputHandler.h"

#include "FieldRenderer.h"

#include "Game.h"

class GameLoop{

public:

GameLoop();

void startGame();

};

Название файла: TerminalInputHandler.cpp

#include "TerminalInputHandler.h"

TerminalInputHandler::TerminalInputHandler(){

loadCommands("commands.txt");

}

Command TerminalInputHandler::getKey(){

std::cout << "Введите ключ команды\n";

char key;

std::cin >> key;

return commands[key];

}

std::tuple<int, int> TerminalInputHandler::getCoords(){

int x,y;

std::cout << "\nВведите координаты (x y): ";

std::cin >> x >> y;

std::tuple<int,int> coords(x,y);

return coords;

}

std::tuple<int, int, int> TerminalInputHandler::getCoordsShip(){

int x,y,orientation;

std::cout << "\nВведите координаты (x y горизонтальность): ";

std::cin >> x >> y >> orientation;

std::tuple<int,int,int> coords(x,y,orientation);

return coords;

}

void TerminalInputHandler::loadCommands(const std::string& filename){

std::ifstream file(filename);

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

// Загрузка команд по умолчанию

commands['s'] = Command::attack;

commands['a'] = Command::ability;

commands['S'] = Command::save;

commands['L'] = Command::load;

return;

}

std::string key, command;

while (file >> key >> command) {

if (commands.find(key[0]) != commands.end()) {

std::cout << "Duplicate key in commands file: " << key << '\n';

}

int opt;

std::string s1="attack";

std::string s2="ability";

std::string s3="save";

std::string s4="load";

if(s1.find(command)!=-1){

commands[key[0]] = Command::attack;

}else if(s2.find(command)!=-1){

commands[key[0]] = Command::ability;

}else if(s3.find(command)!=-1){

commands[key[0]] = Command::save;

}else if(s4.find(command)!=-1){

commands[key[0]] = Command::load;

}else{

std::cout << "Неизвестная команда" << command << '\n';

}

}

}

Название файла: TerminalInputHandler.h

#ifndef TERMINALINPUTHANDLER\_H

#define TERMINALINPUTHANDLER\_H

#include <iostream>

#include <unordered\_map>

#include <fstream>

#include <stdexcept>

#include "GameController.h"

class TerminalInputHandler {

public:

TerminalInputHandler();

Command getKey();

std::tuple<int, int> getCoords();

std::tuple<int, int, int> getCoordsShip();

private:

void loadCommands(const std::string& filename);

std::unordered\_map<char, Command> commands;

//std::unordered\_map<char, std::string> commands;

};

#endif // TERMINALINPUTHANDLER\_H

Название файла: GameRenderer.h

#ifndef GAMERENDERER\_H

#define GAMERENDERER\_H

#include "Game.h"

template <typename Renderer>

class GameRenderer {

public:

GameRenderer(Game& \_game, Renderer& \_renderer) : game(\_game), renderer(\_renderer) {}

void render() {

renderer.render(game.getPlayerField(), game.getEnemyField());

}

void renderPlayer(){

renderer.renderPlayer(game.getPlayerField());

}

void print(const std::string& str){

renderer.print(str);

}

private:

Game& game;

Renderer& renderer;

};

#endif // GAMERENDERER\_H

Название файла: CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(NavalBattle)

set(CMAKE\_CXX\_STANDARD 17)

#aux\_source\_directory(./src tree)

add\_executable(NavalBattle

main.cpp

Ship.cpp

ShipManager.cpp

GameField.cpp

AbilityManager.cpp

InfoHolder.cpp

./Abilities/ShootingAbility.cpp

./Abilities/ShootingAbilityFactory.cpp

./Abilities/ScannerAbility.cpp

./Abilities/ScannerAbilityFactory.cpp

./Abilities/DoubleDamageAbility.cpp

./Abilities/DoubleDamageAbilityFactory.cpp

Game.cpp

Player.cpp

GameState.cpp

GameLoop.cpp

TerminalInputHandler.cpp

FieldRenderer.cpp

)

target\_include\_directories(NavalBattle PUBLIC ${CMAKE\_SOURCE\_DIR})