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 _____ Канцеров А.Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Разработать классы для создания игры типа «Морской бой», реализующие игровую логику, взаимодействие между игроком и противником, сохранение и загрузку состояния игры. Работа направлена на изучение и применение принципов ООП, включая инкапсуляцию, абстракцию и модульность, а также на связывание классов в единую систему для последующего использования в полноценной игре.

Задание.

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.

1. В случае проигрыша пользователь начинает новую игру
2. 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.

Описание класса Game

Класс Game реализует управление игровым процессом игры «Морской бой». Он включает основные механики игры: размещение кораблей, ходы игрока и компьютера, сохранение/загрузку состояния игры и начало нового раунда.

Методы класса Game

start_game(std::string file_name). Загружает сохранённое состояние игры из файла. Принцип работы: использует метод load_game для восстановления всех игровых объектов из сохранённого файла.

start_game(int x, int y, int ship_num, std::vector<int> ship_sizes). Инициализирует новую игру с заданными параметрами поля и кораблей. Принцип работы: создаёт объекты для полей (GameField) пользователя и компьютера. Инициализирует менеджеры способностей (AbilityManager) и кораблей (ShipManager) для обеих сторон. Выполняет случайное размещение кораблей для компьютера с помощью random_placement. Включает режим тумана войны для поля компьютера (fog_war).

add_ship_field(int x, int y, int index, bool orientation). Добавляет корабль на поле пользователя. Принцип работы: создаёт объект ShipPlacement для хранения информации о корабле. Проверяет корректность размещения через метод add_ship менеджера кораблей пользователя. Если размещение успешно, добавляет информацию о размещении в список user_placement_.

comp_turn(). Реализует ход компьютера. Принцип работы: генерирует случайные координаты атаки с использованием библиотеки <random>. Выполняет атаку на поле пользователя методом attack. Возвращает true, если все корабли пользователя уничтожены (end), иначе false.

user_turn(Turn turn, int x, int y, bool& dual). Выполняет ход пользователя (атаку или использование способности). Принцип работы: проверяет корректность координат. Если ход — атака (ATTACK), вызывает метод attack на поле компьютера и проверяет, был ли убит корабль. Если ход — использование способности (ABILITY), вызывает метод use_ability. Если корабль убит,

добавляет случайную способность игроку. Возвращает true, если все корабли компьютера уничтожены (end), иначе false.

new_round(). Инициализирует новый раунд с сохранением состояния игрока. Принцип работы: освобождает память от объектов поля и менеджера кораблей компьютера. Создаёт новые объекты для поля и менеджера кораблей компьютера. Выполняет случайное размещение кораблей компьютера и включает режим тумана войны.

save_game(std::string file_name). Сохраняет текущее состояние игры в файл. Принцип работы: создаёт объект FileManager для записи данных. Сериализует состояние объектов (поля, кораблей, способностей) с помощью методов serialize. Передаёт сохранённое состояние в объект GameState. Записывает состояние в файл.

load_game(std::string file_name). Загружает состояние игры из файла. Принцип работы: создаёт объект FileManager для чтения данных. Читает сохранённое состояние в объект GameState. Восстанавливает игровые объекты из сериализованных данных. Повторно размещает корабли на поле из сохранённых данных.

random_placement(bool who). Выполняет случайное размещение кораблей на поле. Принцип работы: генерирует случайные координаты и ориентацию корабля. Проверяет корректность размещения через метод add_ship. Если размещение невозможно, повторяет попытку. Хранит информацию о размещении в соответствующем списке (comp_placement_ или user_placement_).

get_field(bool who). Возвращает указатель на игровое поле. Принцип работы: проверяет, запрашивается ли поле компьютера (false) или пользователя (true) и возвращает соответствующий указатель.

serialize(bool who). Сериализует данные о размещении кораблей для сохранения. Принцип работы: формирует строку с координатами, ориентацией и индексами кораблей из списков размещения (comp_placement_ или user_placement_).

aserialize(std::string str, bool who). Восстанавливает данные о размещении кораблей из строки. Принцип работы: Парсит строку, извлекая координаты, ориентацию и индексы кораблей, и добавляет их в соответствующий список размещения.

Класс GameState представляет состояние игры и обеспечивает сериализацию (сохранение и загрузку) текущего состояния. Он позволяет хранить информацию о расположении кораблей, состоянии поля, способностях и других игровых данных для обоих игроков (пользователя и компьютера).

Основные методы:

Сеттеры:

set_ship_manager(std::string ships, PlayerType who) — сохраняет состояние кораблей для указанного игрока.

set_game_field(std::string field, PlayerType who) — сохраняет состояние игрового поля для указанного игрока.

set_ability_manager(std::string ability) — сохраняет состояние способностей пользователя.

set_ship_placement(std::string placement, PlayerType who) — сохраняет данные о размещении кораблей для указанного игрока.

Геттеры:

get_ship_manager(PlayerType who) — возвращает состояние кораблей указанного игрока.

get_game_field(PlayerType who) — возвращает состояние игрового поля указанного игрока.

get_ability_manager() — возвращает состояние способностей пользователя.

get_ship_placement(PlayerType who) — возвращает данные о размещении кораблей указанного игрока.

Перегрузка операторов:

operator<< — записывает состояние игры в поток вывода.

operator>> — читает состояние игры из потока ввода.

Поля класса:

comp_field_, user_field_ — строки, представляющие состояние поля компьютера и пользователя.

comp_ships_, user_ships_ — строки, представляющие состояние кораблей компьютера и пользователя.

comp_placement_, user_placement_ — строки, описывающие размещение кораблей.

user_ability_ — строка, представляющая состояние способностей пользователя.

Класс FileManager предназначен для управления чтением и записью данных игрового состояния в файл. Он взаимодействует с классом GameState для сохранения и загрузки текущего состояния игры.

Основные методы:

1. **Конструктор FileManager(std::string file_name, Target target)**:

Принимает имя файла (file_name) и цель (Target) — чтение или запись. Если цель — READ, открывает файл на чтение (std::ifstream). Если цель — WRITE, открывает файл на запись (std::ofstream). Если файл не удается открыть, выбрасывается исключение с текстом "file is bad".

2. **Метод read_state(GameState& state)**:

Использует поток input_file_ для чтения состояния игры из файла. Данные читаются с использованием перегруженного оператора >> класса GameState.

3. **Метод write_state(GameState state)**:

Использует поток output_file_ для записи состояния игры в файл. Данные записываются с использованием перегруженного оператора << класса GameState.

4. **Деструктор ~FileManager()**:

Закрывает файлы, открытые для чтения или записи.

Поля: input_file_ — поток ввода (std::ifstream), используется для чтения данных из файла. output_file_ — поток вывода (std::ofstream), используется для записи данных в файл.

Выводы.

В ходе работы были разработаны и связаны между собой классы, обеспечивающие реализацию основных механик игры типа «Морской бой», включая управление игровым процессом, взаимодействие игрока с противником и сохранение текущего состояния игры. Реализация позволила изучить и применить на практике принципы объектно-ориентированного программирования, такие как инкапсуляция, абстракция и модульность. Полученные результаты создают основу для дальнейшей разработки игры, включая добавление новых механик и улучшение пользовательского интерфейса.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cc

```
#include "./headers/Game.h"
#include <iostream>

void rendering(Game* game) {
  GameField* c = game->get_field(0);
  GameField* u = game->get_field(1);
  int h = c->get_height();
  int w = c->get_width();
  GameField::CellCharacteristics* cell;
  std::cout << "  ";
  for (int i = 1; i < w + 1; ++i) {
    std::cout << i << ' ';
  }
  std::cout << "    ";
  for (int i = 1; i < w + 1; ++i) {
    std::cout << i << ' ';
  }
  std::cout << std::endl;
  for (int y = 1; y < h + 1; ++y) {
    std::cout << y << ' ';
    for (int x = 1; x < w + 1; ++x) {
      cell = c->get_cell(x, y);
      switch (cell->status) {
      case GameField::Status::SEA:
        std::cout << "~ ";
        break;
      case GameField::Status::UNKNOW:
        std::cout << "o ";
        break;
      case GameField::Status::SHIP:
        switch (cell->ship->get_status_segement(cell->segment_ship)) {
        case Ship::Segment::WOUNDED:
          std::cout << "s ";
          break;
        case Ship::Segment::DESTROYED:
```

```
          std::cout << "x ";
          break;
        case Ship::Segment::FULL:
          std::cout << "@ ";
        }
        break;
      }
    }
    std::cout << "    ";
    for (int x = 1; x < w + 1; ++x) {
      cell = u->get_cell(x, y);
      switch (cell->status) {
      case GameField::Status::SEA:
        std::cout << "~ ";
        break;
      case GameField::Status::UNKNOW:
        std::cout << "o ";
        break;
      case GameField::Status::SHIP:
        switch (cell->ship->get_status_segement(cell->segment_ship)) {
        case Ship::Segment::WOUNDED:
          std::cout << "s ";
          break;
        case Ship::Segment::DESTROYED:
          std::cout << "x ";
          break;
        case Ship::Segment::FULL:
          std::cout << "@ ";
        break;
        }
      }
    }
    std::cout << std::endl;
  }
}

int main () {
  bool fl = false;
```

```
while(1) {
  Game *game = new Game();
  std::cout << "load?" << std::endl;
  int ab = 0;
  std::cin >> ab;
  int x = 0, y = 0;
  if (ab) {
    try{
    game->start_game("game.txt");
    } catch (const char* e) {
      std::cout << e << std::endl;
    }
  } else {
    ab = 0;
    int count_ship = 0, size = 0;
    std::vector<int> sizes;
    std::cout << "x y" << std::endl;
    std::cin >> x >> y;
    std::cout << "count of ships: ";
    std::cin >> count_ship;
    for (int i = 0; i < count_ship; ++i) {
      std::cout << i+1 << " ship have size: ";
      std::cin >> size;
      sizes.push_back(size);
    }
    try {
    game->start_game(x, y, count_ship, sizes);
    } catch (const char* e) {
      std::cout << e << std::endl;
      continue;
    }
    int xs = 0, yx = 0, o = 0;
    bool orientation;
    for (int i = 0; i < count_ship; ++i) {
      std::cout << "x y o" << std::endl;
      std::cin >> x >> y >> o;
      if (o) {
        orientation = true;
```

```
    } else {
      orientation = false;
    }
    try {
    game->add_ship_field(x, y, i, orientation);
    } catch (IncorrectPlaceShip& e){
      --i;
      std::cout << e.what() << std::endl;
    }
  }
}
// std::system("cls");
rendering(game);
bool dual = false;
while (true) {
  if (game->comp_turn()) {
    // std::system("cls");
    rendering(game);
    std::cout << "lose" << std::endl;
    std::cout << "new game?" << std::endl;
    std::cin >> ab;
    if (!ab) {
      fl = true;
    }
    break;
  }
  std::cout << "save?" << std::endl;
  std::cin >> ab;
  if (ab) {
    game->save_game("game.txt");
  }
  std::cout << "load?" << std::endl;
  std::cin >> ab;
  if (ab) {
    try{
    game->load_game("game.txt");
    } catch (const char* e) {
      std::cout << e << std::endl;
```

```
  }
}
std::cout << "ability?" << std::endl;
std::cin >> ab;
if (ab) {
  std::cout << "x y" << std::endl;
  std::cin >> x >> y;
  try {
  if (game->user_turn(Game::Turn::ABILITY, x, y, dual)) {
    // std::system("cls");
    rendering(game);
    std::cout << "win" << std::endl;
    std::cout << "new round?" << std::endl;
    std::cin >> ab;
    if (ab) {
      game->new_round();
    } else {
      fl = true;
    }
  }
  } catch (LackAbillity& e) {
    std::cout << e.what() << std::endl;
  } catch (WrongCoordinates& e) {
    std::cout << e.what() << std::endl;
  }
}
// std::system("cls");
rendering(game);
std::cout << "atack\nx y" << std::endl;
std::cin >> x >> y;
try{
if (game->user_turn(Game::Turn::ATTACK, x, y, dual)) {
  // std::system("cls");
  rendering(game);
  std::cout << "win" << std::endl;
  std::cout << "new round?" << std::endl;
  std::cin >> ab;
  if (ab) {
```

```
          game->new_round();
        } else {
          fl = true;
        }
      }
      } catch (WrongCoordinates& e) {
        std::cout << e.what() << std::endl;
      }
      // std::system("cls");
      rendering(game);
    }
    if (fl) {
      break;
    }
  }
}
```

Название файла: ./sources/Game.cpp

```
#include "../headers/Game.h"

auto Game::start_game(std::string file_name) -> void {
  load_game(file_name);
}

auto Game::start_game(int x, int y, int ship_num, std::vector<int>
ship_sizes) -> void {
  comp_field_ = new GameField(x, y);
  user_field_ = new GameField(x, y);
  user_ability_ = new AbilityManager();
  user_ships_ = new ShipManager(ship_num, ship_sizes);
  comp_ships_ = new ShipManager(ship_num, ship_sizes);
  try {
  random_placement(false);
  } catch (const char* e) {
    throw e;
  }
  comp_field_->fog_war();
}
```

```
auto Game::add_ship_field(int x, int y, int index, bool orientation) ->
void {
  GameField::ShipPlacement placement;
  placement.index = index;
  placement.x = x;
  placement.y = y;
  placement.orientation = orientation;
  try{
    user_ships_->add_ship(user_field_, x, y, orientation, index);
  } catch (IncorrectPlaceShip& e) {
    throw e;
  }
  user_placement_.push_back(placement);
}

auto Game::comp_turn() -> bool {
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<> coordX(1, comp_field_->get_width());
  std::uniform_int_distribution<> coordY(1, comp_field_->get_height());
  int x = 0, y = 0;
  x = coordX(gen);
  y = coordY(gen);
  user_field_->attack(x, y, 0);
  return user_ships_->end();
}

auto Game::user_turn(Turn turn, int x, int y, bool& dual) -> bool{\
   if (x > comp_field_->get_width() || x <= 0 || y <= 0 || y >
comp_field_->get_height()) {
    throw WrongCoordinates();
  }
  if (turn == Turn::ATTACK) {
    if (comp_field_->attack(x, y, dual)) {
      user_ability_->add_abiliry_rand();
    }
    dual = false;
```

```
  } else {
    try {
    if (user_ability_->use_ability(x, y, comp_field_, dual)) {
      user_ability_->add_abiliry_rand();
    }
    } catch (LackAbillity& e) {
      throw e;
    }
  }
  return comp_ships_->end();
}

auto Game::new_round() -> void {
  delete comp_field_;
  delete comp_ships_;
   comp_field_  =  new  GameField(user_field_->get_width(),  user_field_-
>get_height());
    comp_ships_   =   new   ShipManager(user_ships_->get_number_ships(),
user_ships_->get_sizes());
  comp_placement_.clear();
  try{
  random_placement(false);
  } catch (const char* e) {
    throw e;
  }
  comp_field_->fog_war();
}

auto Game::save_game(std::string file_name) -> void {
  FileManager file(file_name, FileManager::Target::WRITE);
  state_.set_ability_manager(user_ability_->serialize());
                          state_.set_game_field(user_field_->serialize(),
GameState::PlayerType::PLAYER);
                          state_.set_game_field(comp_field_->serialize(),
GameState::PlayerType::COMPUTER);
                        state_.set_ship_manager(user_ships_->serialize(),
GameState::PlayerType::PLAYER);
```

```
                        state_.set_ship_manager(comp_ships_->serialize(),
GameState::PlayerType::COMPUTER);
                               state_.set_ship_placement(serialize(false),
GameState::PlayerType::COMPUTER);
                                state_.set_ship_placement(serialize(true),
GameState::PlayerType::PLAYER);
  file.write_state(state_);
}

auto Game::load_game(std::string file_name) -> void {
  try {
  FileManager file(file_name, FileManager::Target::READ);
  file.read_state(state_);
                    comp_field_                    =                    new
GameField(state_.get_game_field(GameState::PlayerType::COMPUTER));
                    user_field_                    =                    new
GameField(state_.get_game_field(GameState::PlayerType::PLAYER));
  user_ability_ = new AbilityManager(state_.get_ability_manager());
                    user_ships_                    =                    new
ShipManager(state_.get_ship_manager(GameState::PlayerType::PLAYER));
                    comp_ships_                    =                    new
ShipManager(state_.get_ship_manager(GameState::PlayerType::COMPUTER));
     aserialize(state_.get_ship_placement(GameState::PlayerType::PLAYER),
true);
   aserialize(state_.get_ship_placement(GameState::PlayerType::COMPUTER),
false);
  for (int i = 0; i < comp_ships_->get_number_ships() ; ++i) {
         comp_ships_->add_ship_save(comp_field_,    comp_placement_[i].x,
comp_placement_[i].y,                    comp_placement_[i].orientation,
comp_placement_[i].index);
           add_ship_field(user_placement_[i].x,     user_placement_[i].y,
user_placement_[i].index, user_placement_[i].orientation);
  }
  } catch (...) {
    throw "file is bad.";
  }
}
```

```
auto Game::random_placement(bool who) -> void {
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<> coordX(1, comp_field_->get_width());
  std::uniform_int_distribution<> coordY(1, comp_field_->get_height());
  int x = 0, y = 0;
  bool orientation;
  GameField::ShipPlacement placement;
  if (!who) {
    int count = comp_field_->get_width() * comp_field_->get_height() * 4,
in = 0;
    for (int i = 0; i < comp_ships_->get_number_ships(); ++i) {
      placement.index = i;
      x = coordX(gen);
      y = coordY(gen);
      orientation = coordX(gen) % 2 ? true : false;
      try {
        comp_ships_->add_ship(comp_field_, x, y, orientation, i);
        placement.x = x;
        placement.y = y;
        placement.orientation = orientation;
      } catch (IncorrectPlaceShip& e){
        ++in;
        if (in == count) {
          throw "computer cry.";
        }
        --i;
        continue;
      }
      comp_placement_.push_back(placement);
    }
  }
}

auto Game::get_field(bool who) -> GameField* {
  if (!who) return comp_field_;
  return user_field_;
}
```

```
auto Game::serialize(bool who) -> std::string {
  std::string res;
  if (who) {
    for (int i = 0; i < user_ships_->get_number_ships(); ++i) {
      res += std::to_string(user_placement_[i].index);
      res += " ";
      res += std::to_string(user_placement_[i].orientation);
      res += " ";
      res += std::to_string(user_placement_[i].x);
      res += " ";
      res += std::to_string(user_placement_[i].y);
      res += " ";
    }
  } else {
    for (int i = 0; i < comp_ships_->get_number_ships(); ++i) {
      res += std::to_string(comp_placement_[i].index);
      res += " ";
      res += std::to_string(comp_placement_[i].orientation);
      res += " ";
      res += std::to_string(comp_placement_[i].x);
      res += " ";
      res += std::to_string(comp_placement_[i].y);
      res += " ";
    }
  }
  return res;
}

auto  Game::aserialize(std::string str, bool who) -> void {
  GameField::ShipPlacement placement;
  if (!who) {
    for (int i = 0; i < comp_ships_->get_number_ships(); ++i) {
      placement.index = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      placement.orientation = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      placement.x = std::stoi(str.substr(0, str.find(" ")));
```

```
      str = str.substr(str.find(" ") + 1);
      placement.y = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      comp_placement_.push_back(placement);
    }
  } else {
    for (int i = 0; i < user_ships_->get_number_ships(); ++i) {
      placement.index = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      placement.orientation = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      placement.x = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      placement.y = std::stoi(str.substr(0, str.find(" ")));
      str = str.substr(str.find(" ") + 1);
      user_placement_.push_back(placement);
    }
  }
}
```

Название файла: ./sources/GameState.cpp

```
#include "../headers/GameState.h"

std::ostream& operator<<(std::ostream& out, const GameState& state) {
  out << state.user_ships_ << "\n"
      << state.user_placement_ << "\n"
      << state.user_field_ << "\n"
      << state.user_ability_ << "\n"
      << state.comp_ships_ << "\n"
      << state.comp_placement_ << "\n"
      << state.comp_field_ << "\n";
  return out;
}

std::istream& operator>>(std::istream& in, GameState& state) {
  std::string temp;
  getline(in, temp);
  state.set_ship_manager(temp, GameState::PlayerType::PLAYER);
```

```
  getline(in, temp);
  state.set_ship_placement(temp, GameState::PlayerType::PLAYER);
  getline(in, temp);
  state.set_game_field(temp, GameState::PlayerType::PLAYER);
  getline(in, temp);
  state.set_ability_manager(temp);
  getline(in, temp);
  state.set_ship_manager(temp, GameState::PlayerType::COMPUTER);
  getline(in, temp);
  state.set_ship_placement(temp, GameState::PlayerType::COMPUTER);
  getline(in, temp);
  state.set_game_field(temp, GameState::PlayerType::COMPUTER);
  return in;
}

auto GameState::set_ship_manager(std::string ships, PlayerType who) ->
void {
  if (who == GameState::PlayerType::COMPUTER) {
    comp_ships_ = ships;
  } else {
    user_ships_ = ships;
  }
}

auto GameState::set_game_field(std::string field, PlayerType who) -> void
{
  if (who == GameState::PlayerType::COMPUTER) {
    comp_field_ = field;
  } else {
    user_field_ = field;
  }
}

auto GameState::set_ability_manager(std::string ability) -> void {
  user_ability_ = ability;
}
```

```
auto GameState::set_ship_placement(std::string placement, PlayerType who)
-> void {
  if (who == GameState::PlayerType::COMPUTER) {
    comp_placement_ = placement;
  } else {
    user_placement_ = placement;
  }
}

auto GameState::get_ship_manager(PlayerType who) -> std::string {
  if (who == GameState::PlayerType::COMPUTER) {
    return comp_ships_;
  }
  return user_ships_;
}

auto GameState::get_game_field(PlayerType who) -> std::string {
  if (who == GameState::PlayerType::COMPUTER) {
    return comp_field_;
  }
  return user_field_;
}

auto GameState::get_ability_manager() -> std::string {
  return user_ability_;
}

auto GameState::get_ship_placement(PlayerType who) -> std::string {
  if (who == GameState::PlayerType::COMPUTER) {
    return comp_placement_;
  }
  return user_placement_;
}
```

Название файла: ./sources/FileManager.cpp

```
##include "../headers/FileManager.h"

FileManager::FileManager(std::string file_name, Target target) {
```

```
  if (target == Target::READ) {
    input_file_.open(file_name);
    if (!input_file_.is_open()){
      throw "file is bad";
    }
  } else {
    output_file_.open(file_name);
    if (!output_file_.is_open()){
      throw "file is bad";
    }
  }
}

auto FileManager::read_state(GameState& state) -> void {
  input_file_ >> state;
}

auto FileManager::write_state(GameState state) -> void {
  output_file_ << state;
}

FileManager::~FileManager() {
  input_file_.close();
  output_file_.close();
}
```

Название файла: ./headers/Game.h

```
#ifndef GAME_H
#define GAME_H

#include "AbilityManager.h"
#include "GameState.h"
#include "FileManager.h"
#include "Exceptions.h"

#include <random>

class Game
```

```
{
public:
  enum Turn{
    ABILITY = 0,
    ATTACK = 1
  };
  Game()=default;
  ~Game() = default;
  auto start_game(std::string file_name) -> void;
   auto  start_game(int  x,  int  y,  int  ship_num,  std::vector<int>
ship_sizes) -> void;
  auto add_ship_field(int x, int y, int index, bool orientation) -> void;
  auto comp_turn() -> bool;
  auto user_turn(Turn turn, int x, int y, bool& dual) -> bool;
  auto new_round() -> void;
  auto save_game(std::string file_name) -> void;
  auto load_game(std::string file_name) -> void;
  auto random_placement(bool who) -> void;
  auto get_field(bool who) -> GameField*;
  auto serialize(bool who) -> std::string;
  auto aserialize(std::string str, bool who) -> void;
private:
  GameField *comp_field_, *user_field_;
  ShipManager *comp_ships_, *user_ships_;
  AbilityManager *user_ability_;
  GameState state_;
  std::vector<GameField::ShipPlacement> comp_placement_, user_placement_;
};

#endif
```

Название файла: ./headers/GameState.h

```
#ifndef GAME_STATE_H
#define GAME_STATE_H

#include "ShipManager.h"

#include <fstream>
```

```
class GameState
{
public:
  enum PlayerType {
    PLAYER = 0,
    COMPUTER = 1
  };
  GameState()=default;
  ~GameState()=default;
   friend std::ostream& operator<<(std::ostream& out, const GameState&
state);
  friend std::istream& operator>>(std::istream& in, GameState& state);
  auto set_ship_manager(std::string ships, PlayerType who) -> void;
  auto set_game_field(std::string field, PlayerType who) -> void;
  auto set_ability_manager(std::string ability) -> void;
  auto set_ship_placement(std::string placement, PlayerType who) -> void;
  auto get_ship_manager(PlayerType who) -> std::string;
  auto get_game_field(PlayerType who) -> std::string;
  auto get_ability_manager() -> std::string;
  auto get_ship_placement(PlayerType who) -> std::string;
private:
  std::string comp_field_, user_field_;
  std::string user_ships_, comp_ships_;
  std::string user_placement_, comp_placement_;
  std::string user_ability_;
};

#endif
```

Название файла: ./headers/FileManager.h

```
#ifndef FILE_MANAGER_H
#define FILE_MANAGER_H

#include "GameState.h"

#include <fstream>
#include <iostream>
```

```
class FileManager {
public:
  enum Target {READ = 0,
 WRITE = 1};
private:
  std::ifstream input_file_;
  std::ofstream output_file_;
public:
  FileManager(std::string file_name, Target target);
  ~FileManager();
  auto read_state(GameState& state) -> void;
  auto write_state(GameState state) -> void;
};

#endif
```

Название файла: Makefile

```
all:
    g++   -g   main.cpp   ./sources/Game.cpp   ./sources/Exceptions.cpp
./sources/GameField.cpp   ./sources/ShipManager.cpp   ./sources/Ship.cpp
./sources/FileManager.cpp   ./sources/Shelling.cpp   ./sources/Scan.cpp
./sources/DoubleDamage.cpp                  ./sources/AbilityManager.cpp
./sources/GameState.cpp -o main
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Ввод	game.txt	Комментарий
PS C:\Users\Hyper\Downloads\OOP_NEW\C load? 0 x y 5 5 count of ships: 2 1 ship have size: 2 2 ship have size: 2 x y o 1 1 1 x y o 3 3 1 1 2 3 4 5 1 2 3 4 5 1 o o o o o @ ~ ~ ~ ~ 2 o o o o o @ ~ ~ ~ ~ 3 o o o o o ~ ~ @ ~ ~ 4 o o o o o ~ ~ @ ~ ~ 5 o o o o o ~ ~ ~ ~ ~ save? 1	2 200&200& 0 1 1 1 1 1 3 3 5 5 12222122222221222212222222 102 2 200&200& 0 1 2 4 1 0 3 2 5 5 00000000000000000000000000	ОК

Проверка метода атаки при помощи случайных атак на поле.

Game.txt	Вывод	Комментарий
2 200&200& 0 1 1 1 1 1 3 3 5 5 12222122222221222212222222 102 2 200&200& 0 1 2 4 1 0 3 2 5 5 00000000000000000000000000	load? 1 1 2 3 4 5 1 2 3 4 5 1 o o o o o @ ~ ~ ~ ~ 2 o o o o o @ ~ ~ ~ ~ 3 o o o o o ~ ~ @ ~ ~ 4 o o o o o ~ ~ @ ~ ~ 5 o o o o o ~ ~ ~ ~ ~	ОК

ПРИЛОЖЕНИЕ В
UML-ДИАГРАММА

Рисунок 1 UML-Диаграмма классов