**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".

1. **Метод read\_state(GameState& state)**:

Использует поток input\_file\_ для чтения состояния игры из файла. Данные читаются с использованием перегруженного оператора >> класса GameState.

1. **Метод write\_state(GameState state)**:

Использует поток output\_file\_ для записи состояния игры в файл. Данные записываются с использованием перегруженного оператора << класса GameState.

1. **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 |
|  |  | ОК |

Проверка метода атаки при помощи случайных атак на поле.

|  |  |  |
| --- | --- | --- |
| Game.txt | Вывод | Комментарий |
|  |  | ОК |

**ПРИЛОЖЕНИЕ В**

**UML-ДИАГРАММА**

Рисунок 1 UML-Диаграмма классов

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание