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. 3342 _____ Антипина В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Целью работы является связывание ранее написанных классов в классе игры и создание класса состояния поля, чтобы предоставить пользователю возможность сохранять и загружать игру.

Задание

а) 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.

б) 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 самой игрой можно использовать обертки над командами
- При работе с файлом используйте идиому RAII.

Выполнение работы

В ходе выполнения лабораторной работы были созданы классы Game и GameState — класс состояния игры.

В классе содержатся указатели на объекты класса Gameboard (игровые поля для первого и второго игрока — компьютера), класса ShipManager (менеджеры кораблей пользователя и компьютера), указатель на очередь способностей, а также объект класса InputHandler (обработчик ввода) и двумерные вектора, отвечающие за расстановку кораблей на поле. На данный момент в конструкторе пользователю предлагается загрузить игру. Если он решит загрузить игру, когда сохранений нет, или если он изменил содержимое файла сохранения, программа завершится с сообщением об ошибке. Если этого не произошло, игра начинается, если игра загружена, или запрашивает ввод параметров для инициализации полей класса, если загрузка не нужна.

В классе переопределены операторы ввода и вывода в поток следующим образом:

В поток выводится количество кораблей в менеджере кораблей (у обоих игроков оно совпадает). Затем формируются вектора nums, sizes, содержащие количество кораблей определённого размера и размеры (например, nums = {1, 0, 2, 0 }, sizes = {1, 2, 3, 4} означает, что есть 1 корабль длины 1, 0 кораблей длин 2 и 4, 2 корабля длины 3. Эти вектора в строковой форме выводятся в поток.

Затем для каждого корабля в менеджере сначала игрока, а затем его компьютерного врага выводятся длина корабля и состояние его сегментов. Выводятся размеры поля в строковой форме через пробел. Выводится расстановка кораблей для каждого менеджера в формате <<string>>{x} {y} {pos}. Если на момент сохранения уже производилась атака по пустым клеткам поля, координаты открытых пустых клеток записываются в вектор и выводятся в поток вывода. Для менеджера способностей выводится строка, кодирующая хранящиеся там способности.

Для загрузки: считывается первая строка, преобразуется в число, сохраняется в переменную capacity. Далее считываются строки, содержащие

элементы векторов nums, sizes через пробел в формате строки, разбиваются, приводятся к целочисленному формату, сохраняются в вектора. С помощью них создаются менеджеры кораблей для обоих игроков. Предыдущие менеджеры удаляются. Считываются корабли пользователя и компьютера. Если состояние сегмента - «уничтожен», соответствующему кораблю в менеджере наносится единичный урон, а индекс сегмента сохраняется в специальном векторе по номеру корабля. Если сегмент поврежден, урон ему не наносится, его индекс сразу записывается в вектор. Определяются размеры полей, создаются игровые поля, прежние удаляются. Определяются данные для размещения кораблей, по ним на полях расставляются корабли. Если для размещённого корабля есть индексы в векторе повреждённых сегментов, по нему производится атака (координаты вычисляются в зависимости от ориентации корабля; вызывается метод поля). Также для поля противника сохранялись координаты обнаруженных пустых клеток. По этим клеткам наносится урон уже на новом поле (чтобы для пользователя они оставались видимыми). Для восстановления очереди способностей для каждого символа закодированной строки вызывается метод менеджера способностей, который добавляет способность по её коду.

Загрузка и сохранение реализованы в классах LoadKeeper(), SaveKeeper(), Load(), Save(). Первые два работают непосредственно с файлом сохранения. Открытие/закрытие файла осуществляется в соответствии с RAII. Следующие два класса вычисляют контрольную сумму для файла сохранения, записывают её в файл и сравнивают её значение с вновь вычисленной суммой при загрузке.

Класс Game представляет собой непосредственно игровую сессию. Он содержит поля state: GameState* и методы *public* start(), *private* move(), save(), load(), round(), AddObserver(), RemoveObserver, Notify().

Метод start() реализует начало игры и смену раундов при удачном завершении. Он возвращает 1.

Метод move() реализует ход пользователя. В свой ход пользователь может применить способность (необходимость применения способности запрашивается у игрока, вызывается метод менеджера способностей —

useAbility()), сохраниться, загрузить игру, атаковать противника. Если введены неправильные координаты для атаки, ввод запрашивается до тех пор, пока не будут введены корректные данные (это не касается промахов — только попаданий за пределы поля/отрицательных значений). Компьютер атакует сразу после пользователя случайно.

Метод save() вызывает соответствующий метод у поля state, load() - аналогично. В методе round() вызываются эти методы.

Для вывода в консоль реализован класс ConsolePrinter(), которому игра посылает уведомление о необходимой печати с помощью метода Notify(). В качестве аргумента этому методу передаётся состояние, которое позволяет однозначно определить, что именно необходимо вывести пользователю.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1

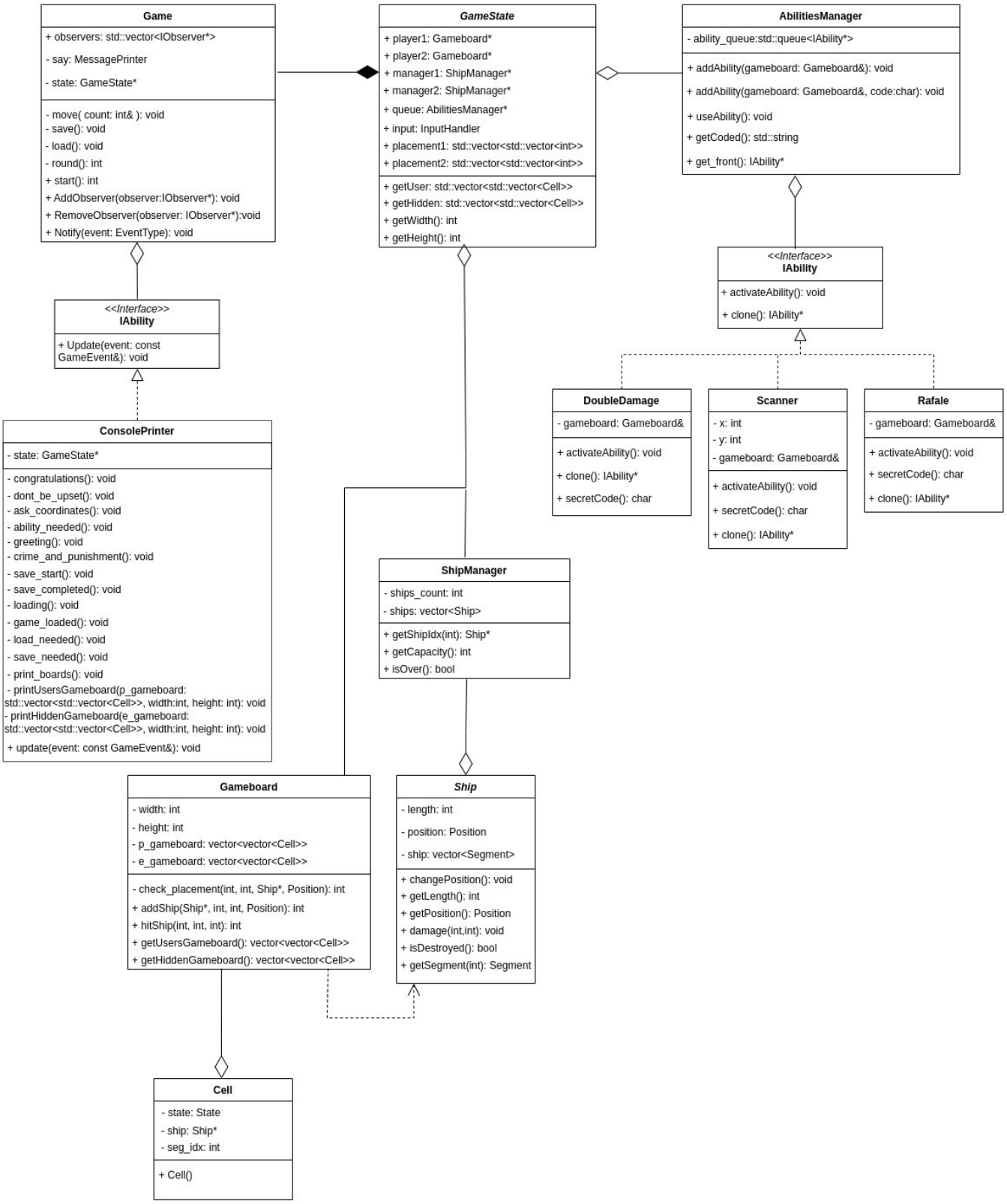

Рисунок 1 – диаграмма классов

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы

Был реализован класс игры, связывающий ранее написанные классы. Были реализованы сохранение и загрузка игры, смена ходов пользователя и компьютера в раунде, смена раундов в игре, начало новой игры при проигрыше.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: game.h

```
#include "time.h"
#include "ability_manager.h"
#include "ship_manager.h"
#include "game_state.h"

class Game{
    MessagePrinter say;
   // InputHandler input;
    GameState* state;

    void
    move( int& count );

    void
    save();

    void
    load();

    int round();
public:
      //Game( Gameboard& player1, Gameboard& player2, ShipManager&
manager1, ShipManager& manager2, AbilitiesManager& queue );
    Game();
   ~Game();
    int start();
};
```

Название файла: game.cpp

```
#include "../headers/game_state.h"
#include "../headers/game.h"
#include "file_not_found.cpp"
#include "cheating_alert.cpp"
//think about it!
void
Game::move( int& count ){
    int x,y = 0;
    int answer = 0;


    std::cout << "Do you want to save the game?\n";
    std::cin >> answer;
    if( answer ){
```

```
            try{
                save();
                load();
            }catch( FileDoesNotExist& e ){
                std::cout << e.what();
            }catch( CheatingAlert& e ){
                std::cout << e.what();
            }
        }

        say.ability_needed();
        try{
            state->input.get_answer( answer );
        }
        catch( IncorrectAbilityAnswer& e ){
            std::cout << e.what();
            say.crime_and_punishment();
            answer = 0;
        }

        if( answer ){
            try{
                state->queue->useAbility();
            }
            catch( EmptyQueue& e ){
                std::cout << e.what();
            }
        }

        say.ask_coordinates();
        bool flag = false;
        do{
            try{
                state->input.get_x_y( x, y );
                state->player2->hitShip( x, y );
                 std::cout << "COUNT: " << count << "ARENOW: " << state-
>manager2->areLeft() << '\n';
                flag = true;
```

```
            }catch(MissingAim& e){
                std::cout << e.what();
                flag = false;
            }
        }while( flag == false );
        state->player2->reset_damage();
        if( state->manager2->areLeft() != count ){
            state->queue->addAbility( *(state->player2) );
            count = state->manager2->areLeft();
        }
    }

    void
    Game::save(){
        state->save();
    }

    void
    Game::load(){
        state->load();
    }

    int
    Game::round(){
        Printer printer;
        //InputHandler input;
        int x = 0;
        int y = 0;
        int count = state->manager2->areLeft();

           while(  state->manager1->isOver()  !=  1  &&  state->manager2-
>isOver() != 1){
            move( count );

            int flag = false;
            time_t timer;
            std::srand(time(&timer));
```

```
            do{
                try{
                    x = std::rand() % (state->player1->getWidth());
                    y = std::rand() % (state->player1->getHeight());
                    state->player1->hitShip( x, y );
                    flag = true;
                }catch(MissingAim& e){
                    flag = false;
                }
            }while( flag == false );

               std::vector < std::vector <Cell> > users_gameboard_1 =
state->player1->getUsersGameboard();
                printer.printUsersGameboard( users_gameboard_1, state-
>player1->getWidth(), state->player1->getHeight() );

              std::vector < std::vector <Cell> > hidden_gameboard_2 =
state->player2->getHiddenGameboard();
               printer.printHiddenGameboard( hidden_gameboard_2, state-
>player2->getWidth(), state->player2->getHeight() );

        }

        if( state->manager2->isOver() == 1 ){
             std::cout << state->manager2->getShipIdx(0)->isDestroyed()
<< '\n';
            return 1;
        }
        return 0;
    }

    // Game::Game( Gameboard& player1, Gameboard& player2, ShipManager&
manager1, ShipManager& manager2, AbilitiesManager& queue ):
    //         player1(player1),player2(player2),
    //         manager1(manager1),manager2(manager2), queue(queue){}

    Game::Game(){
```

```
          // InputHandler input;
          // say.greeting();
          this->state = new GameState();
               //  (this->input).manager_initialize(  *(state->manager1),
*(state->manager2) );
              //  input.gameboard_initialize(  *(state->player1),  *(state-
>manager1), placement1 );
            //  input.e_gameboard_initialize(  *(state->player1),  *(state-
>player2), *(state->manager1), *(state->manager2), placement2 );
            //  AbilitiesManager*  queue  =  new  AbilitiesManager(  *(state-
>player2) );
          // this->queue = queue;
           //this->state = new GameState( &player1, &player2, &manager1,
&manager2, queue, placement1, placement2 );
         // this->state = GameState();
      }

      Game::~Game(){
          //delete queue;
          delete state;
      }

      int
      Game::start(){
          int round_res = 1;
          while( round_res ){
              round_res = round();
              if( round_res ){
                  say.congratulations();
                       (state->input).enemy_s_manager(  *(state->manager1),
*(state->manager2) );
                            (state->input).e_gameboard_initialize(  *(state-
>player1),  *(state->player2),  *(state->manager1),  *(state->manager2),
state->placement2 );
              }
              else{
                  say.dont_be_upset();
                  break;
```

```
            }
        }
        return 1;
    }
```

Название файла: game_state.h

```
    #ifndef STATE
    #define STATE
    #include "gameboard.h"
    #include "ship_manager.h"
    #include "ability_manager.h"
    #include <sstream>
    #include <fstream>

    class GameState{
    public:

        Gameboard* player1;
        Gameboard* player2;
        ShipManager* manager1;
        ShipManager* manager2;
        AbilitiesManager* queue;
        InputHandler input;
        std::vector<std::vector<int>> placement1;
        std::vector<std::vector<int>> placement2;

        //GameState() = default;
        GameState();
             //GameState( Gameboard* player1, Gameboard* player2,
ShipManager* manager1, ShipManager* manager2, AbilitiesManager* queue,
std::vector<std::vector<int>>& placement1, std::vector<std::vector<int>>&
placement2 );
        ~GameState();

           friend std::ostream& operator<<( std::ostream& out, const
GameState& state );
         friend std::istream& operator>>( std::istream& in, GameState&
state );

        int
        getSum();

        void
        save();

        void
        load();
    };
    #endif
```

Название файла: game_state.cpp

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

    #include "file_not_found.cpp"

    #include "cheating_alert.cpp"
```

```
    //  GameState::GameState(  Gameboard*  player1,  Gameboard*  player2,
ShipManager*  manager1,  ShipManager*  manager2,  AbilitiesManager*  queue,
std::vector<std::vector<int>>& placement1, std::vector<std::vector<int>>&
placement2 ):
    //
player1(player1),player2(player2),manager1(manager1),manager2(manager2),q
ueue(queue), placement1(placement1), placement2(placement2){

    //     }

    GameState::GameState(){
        MessagePrinter say;
        this->manager1 = new ShipManager();
        this->manager2 = new ShipManager();
        this->player1 = new Gameboard();
        this->player2 = new Gameboard();
        this->queue = new AbilitiesManager();
        say.greeting();
        say.load_needed();

        int answer = 0;
        input.get_answer( answer );
        if( answer ){
            try{
                this->load();
            }catch( FileDoesNotExist& e ){
                std::cout << e.what();
                exit(0);
            }catch( CheatingAlert& e ){
                std::cout << e.what();
                exit(0);
            }
        }else{
                        (this->input).manager_initialize(   *(manager1),
*(manager2) );
                  input.gameboard_initialize(  *(player1),  *(manager1),
placement1 );
```

```
                input.e_gameboard_initialize(  *(player1),  *(player2),
*(manager1), *(manager2), placement2 );
                                    AbilitiesManager*    queue    =    new
AbilitiesManager( *(player2) );
            delete this->queue;
            this->queue = queue;
        }
    }

    GameState::~GameState(){
        delete this->manager1;
        delete this->manager2;
        delete this->player1;
        delete this->player2;
        delete this->queue;
    }

    std::ostream& operator<<( std::ostream& out, const GameState& state
){
        //ShipManager1&2
        out << std::to_string(state.manager1->getCapacity()) + '\n';
        std::string nums = "";
        std::string sizes = "";
        Ship* ship1 = state.manager1->getShipIdx(0);
        int length = ship1->getLength();
        int count = 0;

        for( int i = 0; i < state.manager1->getCapacity(); ++i ){
                ship1 = state.manager1->getShipIdx(i);

                if( length != ship1->getLength() ){
                    nums += std::to_string(count) + ' ';
                    count = 0;
                    sizes += std::to_string(length) + ' ';
                    length = ship1->getLength();
                }
                length = ship1->getLength();
                count++;
```

```
        }
        nums += std::to_string(count) + ' ';
        length = ship1->getLength();
        sizes += std::to_string(length) + ' ';
        out << nums + '\n';
        out << sizes + '\n';


        for( int i = 0; i < state.manager1->getCapacity(); ++i ){
            Ship* ship1 = state.manager1->getShipIdx(i);
            out << std::to_string(ship1->getLength()) << '\n';
            for( int j = 0; j < ship1->getLength(); ++j ){
                out << std::to_string(int(ship1->getSegment( j )));
            }
            out << ' ';
            Ship* ship2 = state.manager2->getShipIdx(i);
            for( int j = 0; j < ship2->getLength(); ++j ){
                out << std::to_string(int(ship2->getSegment( j )));
            }
            out << '\n';
        }
        //Gameboard1
           out  <<  std::to_string(state.player1->getWidth())  +  '  '  +
std::to_string(state.player1->getHeight()) + '\n';
        for( int i = 0; i < state.placement1.size(); ++i ){
             out << std::to_string(state.placement1.at(i).at(0)) + ' '
+std::to_string(state.placement1.at(i).at(1))     +     '     '     +
std::to_string(state.placement1.at(i).at(2)) + '\n';
        }


        std::vector<std::vector<int>> empties;
        int vec_i = 0;
             std::vector<std::vector<Cell>>  hidden  =  state.player2-
>getHiddenGameboard();
        for( int j = 0; j < state.player2->getHeight(); ++j ){
            for( int i = 0; i < state.player2->getWidth(); ++i ){
                if( hidden.at(j).at(i).state == State::EMPTY ){
                    empties.resize(vec_i + 1);
                    empties.at(vec_i).resize(2);
```

```
                    empties.at(vec_i++) = { i, j };
                }
            }
        }
        out << std::to_string(vec_i) + '\n';
        for( int v = 0; v < vec_i; ++v ){
                out << std::to_string( empties.at(v).at(0) ) + ' ' +
std::to_string( empties.at(v).at(1) ) << '\n';
        }

        //Gameboard2
        for( int i = 0; i < state.placement2.size(); ++i ){
            out << std::to_string(state.placement2.at(i).at(0)) + ' ' +
std::to_string(state.placement2.at(i).at(1))     +     '     '     +
std::to_string(state.placement2.at(i).at(2)) + '\n';
        }
        //AbilityManager
        out << state.queue->getCoded() + '\n';
        return out;
    }
    std::istream& operator>>( std::istream& in, GameState& state ){
        //ShipManager1&2
        std::string unsplitted;
        std::getline(in, unsplitted);
        int capacity = std::stoi(unsplitted);

        std::vector<int> nums;
        std::vector<int> sizes;

        std::getline(in, unsplitted);
        std::istringstream f(unsplitted);
        while( getline( f, unsplitted, ' ' ) ){
            nums.push_back(std::stoi(unsplitted));
        }
        std::getline(in, unsplitted);
        std::istringstream g(unsplitted);
        while( getline( g, unsplitted, ' ' ) ){
            sizes.push_back(std::stoi(unsplitted));
```

```
}

delete state.manager1;
delete state.manager2;

state.manager1 = new ShipManager( nums, sizes );
state.manager2 = new ShipManager( nums, sizes );

int n;
std::string users;
std::string robo;
std::vector<std::string> un = {"",""};
std::vector<std::vector<int>> indexes_to_hit;
std::vector<std::vector<int>> indexes_to_hit_me;
indexes_to_hit.resize(capacity);
indexes_to_hit_me.resize(capacity);

for( int i = 0; i < capacity; ++i ){
    un.erase( un.begin(), un.end() );
    std::getline(in, unsplitted);//n
    n = std::stoi(unsplitted);

    std::getline(in, unsplitted);
    std::istringstream a(unsplitted);
    while( getline( a, unsplitted, ' ' ) ){
        un.push_back(unsplitted);
    }
    users = un[0];
    robo = un[1];

    for( int j = 0; j < n; ++j ){
        if( users[j] != '0' ){
            if( users[j] == '2' )
                state.manager1->getShipIdx(i)->damage( j, 1 );
            indexes_to_hit_me.at(i).push_back(j);
        }
        if( robo[j] != '0' ){
```

```
                if( robo[j] == '2' )
                    state.manager2->getShipIdx(i)->damage( j, 1 );
                indexes_to_hit.at(i).push_back(j);
            }
        }
    }
    //Gameboard1&2
    int width = 5;
    int height = 5;
    std::vector<int> nu;
    std::getline(in, unsplitted);

    std::istringstream e(unsplitted);
    while( getline( e, unsplitted, ' ' ) ){
        nu.push_back(std::stoi(unsplitted));
    }
    if( nu.size() ){
        width = nu[0];
        height = nu[1];
    }else{
        throw IncorrectBoardSize();
    }

    delete state.player1;
    delete state.player2;

    state.player1 = new Gameboard( height, width );
    state.player2 = new Gameboard( height, width );

     state.placement1.erase(state.placement1.begin(),state.placement
1.end());
    for( int i = 0; i < capacity; i++ ){
        int x, y, pos;
        std::vector<int> u;
        // in >> x >> y >> pos;
        std::getline(in, unsplitted);
        std::istringstream h(unsplitted);
        while( getline( h, unsplitted, ' ' ) ){
```

```
                u.push_back(std::stoi(unsplitted));
            }
            if( u.size() >= 3 ){
                x = u[0];
                y = u[1];
                pos = u[2];
            }
              state.player1->addShip( state.manager1->getShipIdx(i), x,
y, Position(pos) );
            state.placement1.push_back({x,y,pos});
            if( indexes_to_hit_me.at(i).size() ){
                for( int idx = 0; idx < indexes_to_hit_me.at(i).size();
++idx ){
                    if( pos )
                                          state.player1->hitShip(  x  +
indexes_to_hit_me.at(i).at(idx), y );
                    else
                                       state.player1->hitShip(  x,  y  +
indexes_to_hit_me.at(i).at(idx) );
                }
            }
        }

        std::getline(in, unsplitted);
        int vec_i = std::stoi(unsplitted);
        for( int v = 0; v < vec_i; ++v ){
            int x, y;
            std::getline(in, unsplitted);
            std::istringstream h(unsplitted);
            getline(h, unsplitted, ' ');
            x = std::stoi(unsplitted);
            getline(h, unsplitted, ' ');
            y = std::stoi(unsplitted);
            state.player2->hitShip( x, y );
        }

         state.placement2.erase(state.placement2.begin(),state.placement
2.end());
```

```
        for( int i = 0; i < capacity; i++ ){
            int x, y, pos;
            std::vector<int> u;
            // in >> x >> y >> pos;
            std::getline(in, unsplitted);
            std::istringstream h(unsplitted);
            while( getline( h, unsplitted, ' ' ) ){
                u.push_back(std::stoi(unsplitted));
            }
            if( u.size() >= 3 ){
                x = u[0];
                y = u[1];
                pos = u[2];
            }

              state.player2->addShip( state.manager2->getShipIdx(i), x,
y, Position(pos) );
            state.placement2.push_back({x,y,pos});

            if( indexes_to_hit.at(i).size() ){
                 for( int idx = 0; idx < indexes_to_hit.at(i).size(); +
+idx ){
                    if( pos )
                                          state.player2->hitShip(  x  +
indexes_to_hit.at(i).at(idx), y );
                    else
                                       state.player2->hitShip(  x,  y  +
indexes_to_hit.at(i).at(idx) );
                }
            }
        }
        //Abilities
        std::string secret_coded_queue;
        AbilitiesManager q = AbilitiesManager();

        *(state.queue) = q;
        std::getline(in, secret_coded_queue);
        for( int i = 0; i < secret_coded_queue.size(); ++i ){
```

```
                            state.queue->addAbility(   *(state.player2),
secret_coded_queue[i] );
          }



          return in;
      }

      int
      GameState::getSum(){
          int sum = 0;
          std::ifstream file("seabattle.txt");
          std::string line;
          if( !file.is_open() ){
              throw FileDoesNotExist();
          }
          while( std::getline(file, line) )
          {
              std::getline(file, line);
              for( char num: line ){
                  sum += int(num)*2;
              }
          }

          file.close();
          return sum;
      }

      void
      GameState::save(){
          MessagePrinter say;
          say.save_start();
          std::ofstream out;
          out.open("seabattle.txt");      // открываем файл для записи
          if (out.is_open())
          {
              out << *this;
```

```
    }
    out.close();
    std::ofstream strict;
    strict.open("check.txt");
    if( strict.is_open()){
        int control_sum = getSum();
        strict << control_sum;
    }else{
        exit(0);
    }
    strict.close();
    say.save_completed();
}

void
GameState::load(){
    MessagePrinter say;
    say.loading();
    int current_sum = getSum();
    int correct = 0;
    std::ifstream strict;
    strict.open("check.txt");
    if( strict.is_open() ){
        strict >> correct;
    }
    strict.close();
    if( correct != current_sum ){
        throw CheatingAlert();
    }



    std::ifstream in("seabattle.txt"); // окрываем файл для чтения
    if (in.is_open())
    {
        in >> *this;
    }else{
        throw FileDoesNotExist();
    }
```

```
    in.close();
    say.game_loaded();
}
```

Название файла: main.cpp

```
#include "game.h"

int main(){
    int res = 1;
    while( res ){
        Game new_game;
        res = new_game.start();
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Hola chico nuevo!

Or do we have a sea wolf here?

Perhaps it's time to find out!

Time to scrub the deck, tie some sailor knots, and catch a fair wind!

Good luck, sailor!

Do you want to load game? (1/0)

1

Loading...

Load is done!

Do you want to save the game?

1

Saving in process!

Saved!

Loading...

Load is done!

Do you want to use ability? 1/0

1

QUEUE SIZE 3

Ready, aim, fire!

Where do you want to hit?

0 1

COUNT: 1ARENOW: 1

$ *

* *

```
  0 1
0 ? ?
1   $
```

```
Do you want to save the game?
1
Saving in process!
Saved!
Loading...
Load is done!
Do you want to use ability? 1/0
1
QUEUE SIZE 2
Which area do you want to scan?
1 1
Ship is here!
Where do you want to hit?
1 1
COUNT: 1ARENOW: 0
$ *
* *
```

```
  0 1
0 ? ?
1   X
```

```
1
```

Congratulations! Now let's continue that adventure! Next round!

Do you want to save the game?

1

Saving in process!

Saved!

Loading...

Load is done!

Do you want to use ability? 1/0

0

Where do you want to hit?

0 1

COUNT: 1ARENOW: 1

$ *

* *

---

  0 1

0 ? ?

1   ?

Do you want to save the game?

0

Do you want to use ability? 1/0

1

QUEUE SIZE 2

Double damage is activated! Don't miss!

Where do you want to hit?

1 0

COUNT: 1ARENOW: 0

X *

* *

---

0 1

0 ? X

1 ?

1

Congratulations! Now let's continue that adventure! Next round!

Errr.... Better luck next time?

Hola chico nuevo!

Or do we have a sea wolf here?

Perhaps it's time to find out!

Time to scrub the deck, tie some sailor knots, and catch a fair wind!

Good luck, sailor!

Do you want to load game? (1/0)

1

Loading...

Load is done!

Do you want to save the game?

0

Do you want to use ability? 1/0

0

Where do you want to hit?

0 0

COUNT: 1ARENOW: 1

$ *

* *

---

0 1

0 ?

1 ? ?

Do you want to save the game?

0

Do you want to use ability? 1/0

0

Where do you want to hit?

0 0

COUNT: 1ARENOW: 1

$ *

* *

---

0 1

0 ?

1 ? ?

Do you want to save the game?

0

Do you want to use ability? 1/0

0

Where do you want to hit?

0 0

COUNT: 1ARENOW: 1

$ *

* *

---

 0 1

0 ?

1 ? ?

Do you want to save the game?

0

Do you want to use ability? 1/0

1

QUEUE SIZE 2

Double damage is activated! Don't miss!

Where do you want to hit?

0 1

COUNT: 1ARENOW: 1

$ *

* *

---

 0 1

0 ?

1 ?

Do you want to save the game?

1

Saving in process!

Saved!

Loading...

Load is done!

Do you want to use ability? 1/0

Do you want to use ability? 1/0

1

QUEUE SIZE 1

Ready, aim, fire!

Where do you want to hit?

1 1

COUNT: 1ARENOW: 1

$ *

* *

---

 0 1

0 $

1

Do you want to save the game?

0

Do you want to use ability? 1/0

0

Where do you want to hit?

0 0

COUNT: 1ARENOW: 1

$ *

* *

```
  0 1
0   $
1
--------------------------------------------------------------------

Do you want to save the game?
0
Do you want to use ability? 1/0
0
Where do you want to hit?
0 0
COUNT: 1ARENOW: 1
$ *
* *
--------------------------------------------------------------------

  0 1
0   $
1
--------------------------------------------------------------------

Do you want to save the game?
0
Do you want to use ability? 1/0
0
Where do you want to hit?
0 0
COUNT: 1ARENOW: 1
$ *
```

* *

---

 0 1

0 $

1

Do you want to save the game?

0

Do you want to use ability? 1/0

0

Where do you want to hit?

0 0

COUNT: 1ARENOW: 1

X *

* *

---

 0 1

0 $

1

Errr.... Better luck next time?

Hola chico nuevo!

Or do we have a sea wolf here?

Perhaps it's time to find out!

Time to scrub the deck, tie some sailor knots, and catch a fair wind!

Good luck, sailor!

Do you want to load game? (1/0)

Таким образом демонстрируется запуск игры, последовательная смена раундов при победе игрока и перезапуск игры при его поражении, возможность сохранить игру и загрузить, использовать способности.

Последний файл сохранения:

1

1

1

1

1 0

2 2

0 0 0

2

0 0

0 1

1 0 1

2

Контрольная сумма: 416333634