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. 3341 Перевалов П.И.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы

Реализовать классы игрового процесса и сохранения игры.

Задание

Создать класс игры, который реализует следующий игровой цикл:

- Начало игры
- Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
- В случае проигрыша пользователь начинает новую игру
- В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

- Класс игры может знать о игровых сущностях, но не наоборот
- Игровые сущности не должны сами порождать объекты состояния
- Для управления самое игрой можно использовать обертки над командами
- При работе с файлом используйте идиому RAII.

Выполнение работы

В процессе продолжения разработки игры "Морской бой" были реализованы классы для организации игрового процесса: Game, GameState.

Класс GameState

Этот класс был разработан для управления игровыми полями, кораблями и способностями игроков, обеспечивая их сохранение и восстановление из файлов. Рассмотрим методы класса GameState и их алгоритмы.

Конструктор GameState

Конструктор принимает указатели на игровые поля, менеджеры способностей и кораблей игроков и инициализирует их в соответствующих полях объекта.

Алгоритм:

- Сохраняет указатели на игровые поля (игрока и врага), менеджеры способностей и кораблей в приватных переменных объекта.

Метод save

Отвечает за сохранение текущего состояния игры в бинарный файл.

Алгоритм:

- Открывает файл для записи в бинарном режиме.
- Проверяет успешность открытия файла.
- Сохраняет размеры игрового поля игрока.
- Вызывает методы для сохранения состояния полей (save_player_field и save_enemy_field).
- Сохраняет информацию о кораблях игроков (save_player_ships и save_enemy_ships).
- Сохраняет список способностей игрока, записывая их имена.

Метод load

Загружает состояние игры из файла, восстанавливая состояние полей, кораблей и способностей.

Алгоритм:

- Открывает файл для чтения в бинарном режиме.
- Проверяет успешность открытия файла.
- Читает размеры игрового поля.
- Очищает текущие состояния полей.
- Вызывает методы для загрузки состояния полей (load_player_field и load_enemy_field).
- Восстанавливает состояние кораблей игроков (load_player_ships и load_enemy_ships).
- Пересвязывает корабли с клетками поля.
- Загружает способности игрока через метод load_player_abilities.

Методы для сохранения и загрузки полей

save_player_field и save_enemy_field

Сохраняют состояние клеток игрового поля в файл.

Алгоритм:

- Итерируются по каждой клетке игрового поля.
- Записывают состояние клетки (cell_state), индекс корабля (ship_index) и ориентацию корабля (vertical_orientation).
- После записи строки делают переход на новую строку.

load_player_field и load_enemy_field

Загружают состояние клеток игрового поля из файла.

Алгоритм:

- Последовательно читают данные клетки.
- Заполняют состояние, индекс корабля и ориентацию клетки на поле.

Методы для сохранения и загрузки кораблей

save_player_ships и save_enemy_ships

Сохраняют параметры каждого корабля (длину, состояние, идентификатор и состояние HP).

Алгоритм:

- Записывают количество кораблей.

- Итерируются по каждому кораблю, записывая:
- Длину (length).
- Состояние (condition).
- Идентификатор (id).
- Состояние каждой части корпуса (hp_bar).

load_player_ships и load_enemy_ships

Загружают параметры кораблей из файла.

Алгоритм:

- Читают количество кораблей.
- Для каждого корабля восстанавливают:
- Длину, состояние, идентификатор.
- Состояние корпуса, записывая данные в массив hp_bar.

Методы для работы со способностями

save_player_abilities

Сохраняет список способностей игрока в файл, записывая их имена.

Алгоритм:

- Записывает количество способностей.
- Итерируется по способностям игрока, определяя их тип с помощью typeid и записывая имя типа в файл.

load_player_abilities

Загружает способности игрока на основе имен, сохраненных в файле.

Алгоритм:

- Читает количество способностей.
- Удаляет существующие способности игрока.
- По очереди считывает имя способности и добавляет соответствующую способность с помощью менеджера способностей.

Класс Game

Класс реализует размещение кораблей, ходы игрока и врага, проверку состояния игры, управление сохранением и загрузкой, а также контроль игрового цикла. Рассмотрим основные методы и их алгоритмы.

Конструктор Game

Инициализирует основные объекты, необходимые для игры, такие как поля, менеджеры кораблей, способностей, состояние игры и отслеживание текущего раунда.

Алгоритм:

- Устанавливает начальное значение для переменной round (равное 1).
- Инициализирует указатели на:

1. Поля игрока и противника.
2. Менеджеры кораблей игрока и противника.
3. Менеджер способностей игрока.
4. Состояние игры (GameState).

Метод place_ships_randomly

Размещает корабли на игровом поле случайным образом, избегая конфликтов с другими объектами.

Алгоритм:

- Очищает игровое поле (clear_field) и восстанавливает менеджер кораблей (restore_manager).
- Итерируется по каждому кораблю:
- Генерирует случайные координаты и ориентацию (вертикальная или горизонтальная).
- Проверяет, помещается ли корабль в указанные координаты.
- Пытается разместить корабль с помощью метода place_ship.
- Если размещение не удалось, повторяет попытку.

Метод enemy_turn

Реализует логику хода противника.

Алгоритм:

- Генерирует случайные координаты для атаки, проверяя, что клетка не является пустой или уже уничтоженной.
- Печатает координаты атаки.
- Атакует клетку с помощью метода attack_cell и возвращает результат.

Метод user_turn

Реализует логику хода игрока.

Алгоритм:

- Отображает текущие игровые поля:
- Поле игрока (display_your_playing_field).
- Поле противника (display_playing_field_for_enemy).
- Спрашивает пользователя, хочет ли он сохранить игру:
- Если да, сохраняет игру с помощью метода save объекта GameState.
- Осуществляет атаку с использованием способностей игрока (ability_attack_cell).
- Если во время атаки возникает исключение, выводит его сообщение.

Метод is_game_over

Проверяет, остались ли на поле целые или поврежденные корабли.

Алгоритм:

Итерируется по всем клеткам игрового поля.

- Если находит клетку с состоянием UNDAMAGED_SHIP или DAMAGED_SHIP, возвращает false.
- Если ни одной такой клетки не найдено, возвращает true.

Метод start

Управляет запуском игры, включая возможность загрузки сохранения или начала новой партии.

Алгоритм:

- Спрашивает у пользователя, хочет ли он загрузить сохранение.
- Если да, загружает игру с помощью метода load объекта GameState и устанавливает флаг game_loaded в true.

- Запускает игровой цикл:
- Если загружено сохранение, игра продолжается с текущего состояния.
- Если игра начинается с начала:

1. В первом раунде размещает корабли игрока случайным образом.
2. Размещает корабли противника на каждом этапе.
3. Запускает метод play_round, который определяет результат текущего раунда.

Метод play_round

Реализует игровой процесс в рамках одного раунда.

Алгоритм:

- Выводит сообщение о начале раунда.
- Организует чередующиеся ходы игрока и противника:
- Игрок делает ход, используя метод user_turn.
- Если противник больше не имеет кораблей (is_game_over), выводится сообщение о победе, и раунд завершается.
- Противник делает ход, используя метод enemy_turn.
- Если у игрока больше не осталось кораблей, выводится сообщение о поражении, и игра завершается.
- В случае окончания раунда увеличивает счетчик раундов (round).

Архитектурные решения

Класс GameState

Назначение: Отвечает за сохранение и восстановление состояния игры. Основной акцент сделан на сериализации и десериализации объектов.

Архитектурные решения:

- Инкапсуляция состояния: Использует поля для хранения указателей на игровые элементы (поля, менеджеры кораблей, способностей).
- Сериализация/десериализация: Методы save и load делят процесс на отдельные этапы для работы с игровыми полями, кораблями и способностями.

- Гибкость форматов: Состояние сохраняется в бинарном файле с использованием стандартных потоков ввода/вывода C++ (std::ofstream, std::ifstream).
- Модульность: Методы разделены по областям ответственности — работа с полями, кораблями, способностями.

Преимущества:

- Четкое разделение логики.
- Простота расширения для добавления новых компонентов игры.

Класс Game

Назначение: Управляет игровым процессом, включая игровой цикл, размещение кораблей, ходы игроков и проверку окончания игры.

Архитектурные решения:

- Организация игрового цикла: Методы start и play_round структурируют игровой процесс, поддерживая возможность сохранения и загрузки.
- Разделение логики ходов: Методы user_turn и enemy_turn обрабатывают действия игроков отдельно, включая отображение состояния игры и выполнение атак.
- Случайное размещение: Метод place_ships_randomly генерирует координаты с учетом валидности позиции для кораблей.
- Интеграция с GameState: Взаимодействует с классом GameState для загрузки и сохранения игры, минимизируя дублирование кода.

Преимущества:

- Управление игровым процессом централизовано.
- Логика хорошо разделена по методам.
- Поддерживается сохранение и продолжение игры.

Взаимодействие классов

Класс Game использует GameState для управления состоянием игры, предоставляя модульное и легко расширяемое решение:

- GameState отвечает за сохранение/загрузку.
- Game реализует логику игрового процесса.

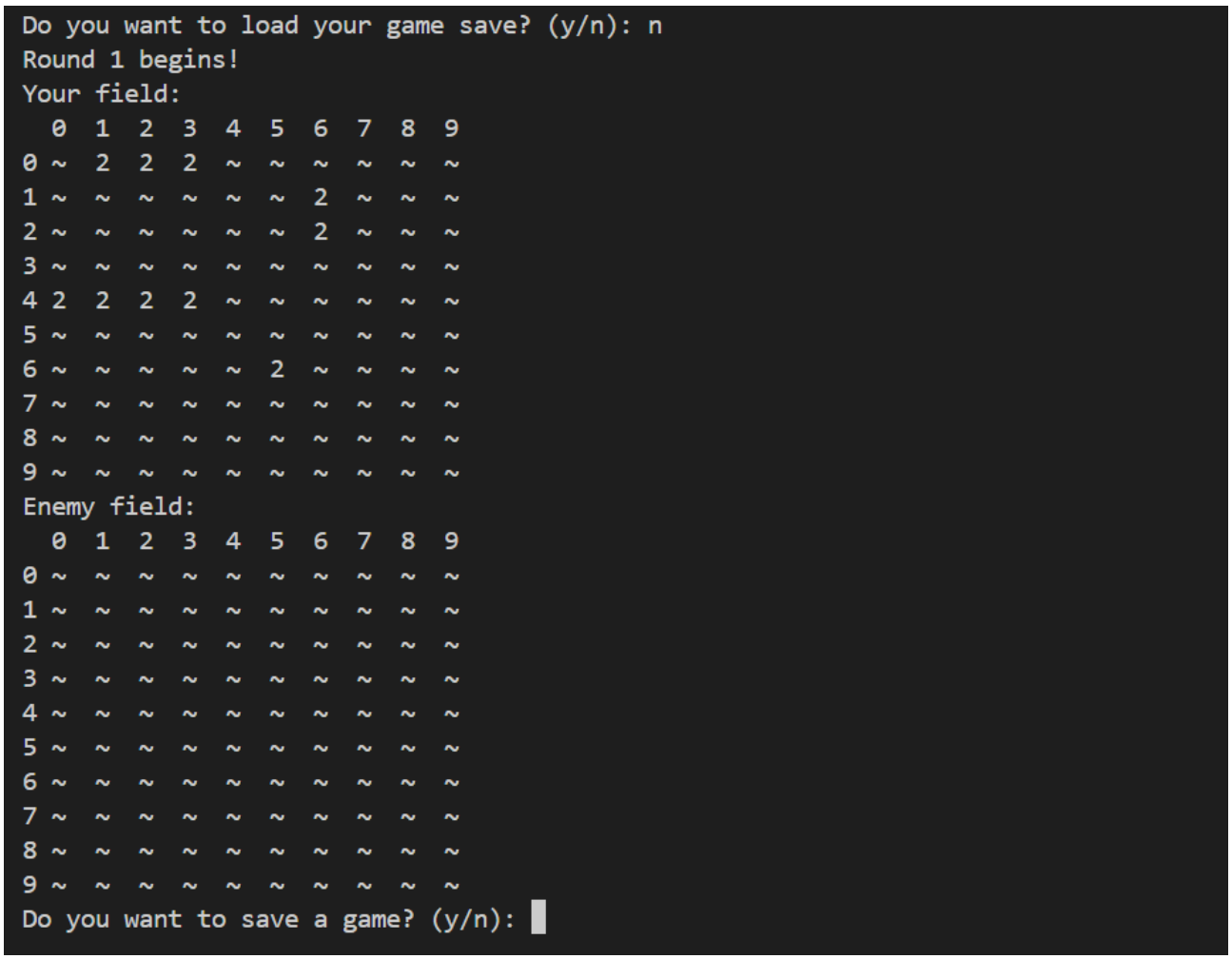
Такое разделение упрощает тестирование, поддержку и добавление новых функций.

```
norton@LAPTOP-563HCLJQ:/mnt/c/Users/79969/Desktop/workspace/sem3/oop/sea_battle/src$ make
g++ -c main.cpp
g++ -c ship.cpp
g++ -c field_cell.cpp
g++ -c ship_manager.cpp
g++ -c sea_battle_playground.cpp
g++ -c exceptions.cpp
g++ -c ability.cpp
g++ -c double_damage.cpp
g++ -c random_bombardment.cpp
g++ -c scanner.cpp
g++ -c ability_manager.cpp
g++ -c game_state.cpp
g++ -c game.cpp
g++ main.o ship.o field_cell.o ship_manager.o sea_battle_playground.o exceptions.o ability.o double_damage.o random_bombardmen
t.o scanner.o ability_manager.o game_state.o game.o -o game
rm *.o
```

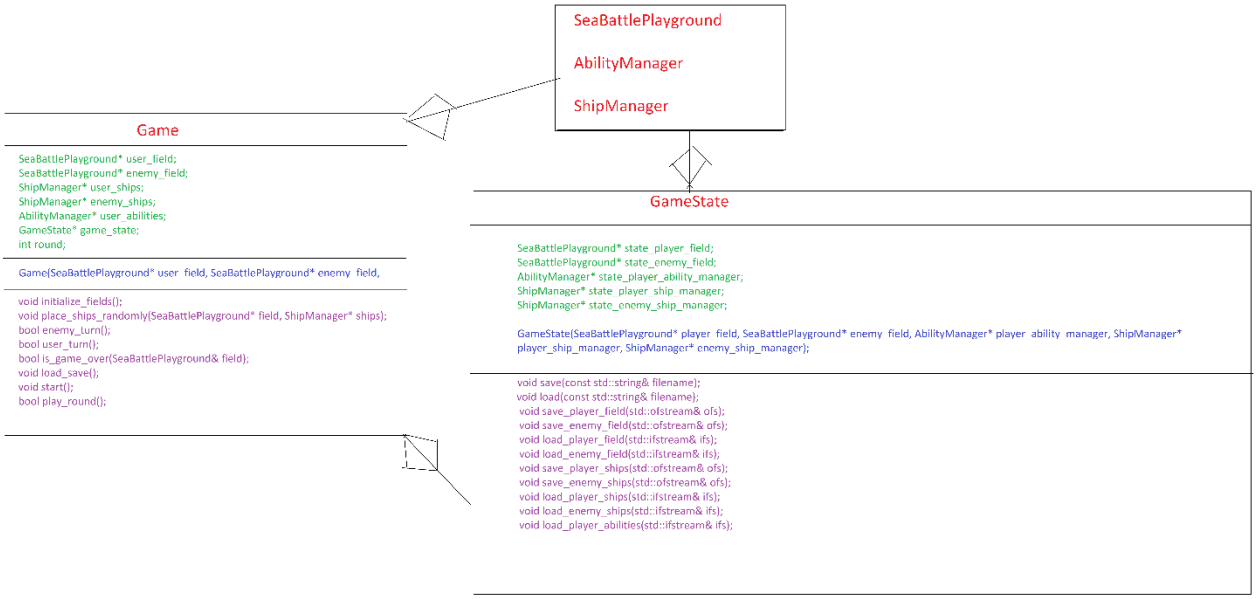
```cpp
#include <iostream>
#include <exception>
#include <vector>
#include <ctime>
#include "sea_battle_playground.h"
#include "ship_manager.h"
#include "ability_manager.h"
#include "game_state.h"
#include "game.h"

int main(){
    std::srand(std::time(0));
    try{
        int field_width = 10;
        int field_length = 10;
        std::vector<int> ship_sizes = {1, 2, 3, 4};

        SeaBattlePlayground user_field(field_width, field_length);
        SeaBattlePlayground enemy_field(field_width, field_length);
        ShipManager user_ships(ship_sizes.size(), ship_sizes);
        ShipManager enemy_ships(ship_sizes.size(), ship_sizes);
        AbilityManager user_abilities;
        GameState game_state(&user_field, &enemy_field, &user_abilities, &user_ships, &enemy_ships);
        Game sea_battle(&user_field, &enemy_field, &user_ships, &enemy_ships, &user_abilities, &game_state);
        sea_battle.start();
    }
    catch (const std::exception& e){
        std::cout << e.what() << std::endl;
    } catch (const char* e) {
        std::cerr << e << std::endl;
    }
    return 0;
}
```



UML диаграмма классов отображена ниже. На ней зеленым цветом отображены поля классов, синим цветом конструкторы/деструкторы, а фиолетовым – методы. Также на диаграмме отображены связи между классами.



Разработанный программный код см. в приложении А.

Выводы

В ходе разработки были созданы классы, которые помогли организовать игровой процесс.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <exception>
#include <vector>
#include <ctime>
#include "sea_battle_playground.h"
#include "ship_manager.h"
#include "ability_manager.h"
#include "game_state.h"
#include "game.h"

int main(){
    std::srand(std::time(0));
    try{
        int field_width = 10;
        int field_length = 10;
        std::vector<int> ship_sizes = {1, 2, 3, 4};

        SeaBattlePlayground user_field(field_width, field_length);
        SeaBattlePlayground enemy_field(field_width, field_length);
        ShipManager user_ships(ship_sizes.size(), ship_sizes);
        ShipManager enemy_ships(ship_sizes.size(), ship_sizes);
        AbilityManager user_abilities;
        GameState game_state(&user_field, &enemy_field,
&user_abilities, &user_ships, &enemy_ships);
        Game sea_battle(&user_field, &enemy_field, &user_ships,
&enemy_ships, &user_abilities, &game_state);
        sea_battle.start();
    }
    catch (const std::exception& e){
        std::cout << e.what() << std::endl;
    } catch (const char* e) {
        std::cerr << e << std::endl;
    }
    return 0;
}
```

Название файла: game.cpp

```
#include <vector>
#include <time.h>
#include "game.h"
#include "game_state.h"

enum CellState {
    EMPTY_CELL = '.',
    UNKNOWN_CELL = '~',
    DAMAGED_SHIP = '1',
    UNDAMAGED_SHIP = '2',
    DESTROYED_SHIP = 'x'
};

void Game::place_ships_randomly(SeaBattlePlayground* field,
ShipManager* ships) {
        field->clear_field();
        ships->restore_manager();
        for (int i = 0; i < ships->quantity; ++i) {
            bool placed = false;
            time_t counter = 0;
            while (!placed) {
                srand(static_cast<unsigned int>(time(&counter)));
                int x = rand() % field->width;
                int y = rand() % field->length;
                bool vertical = rand() % 2;

                if((vertical)&&(y + ships->ships[i].length >
field->length)) continue;
                else{
                    if(x + ships->ships[i].length > field->width)
continue;
                }

                try {
                    field->place_ship(ships->ships[i], x, y,
vertical, i, true);
                    placed = true;
```

```
                    } catch (...) {
                        counter++;
                        continue;
                    }
                }
            }
        }

        bool Game::enemy_turn() {
            int x, y;
            do {
                x = rand() % user_field->width;
                y = rand() % user_field->length;
            } while (user_field->field[x][y].cell_state == EMPTY_CELL ||
                        user_field->field[x][y].cell_state          ==
DESTROYED_SHIP);

            std::cout << "Enemy attacks (" << x << ", " << y << ")" <<
std::endl;
            return user_field->attack_cell(x, y);
        }

        bool Game::user_turn() {
            user_field->display_your_playing_field();
            enemy_field->display_playing_field_for_enemy();

            std::cout << "Do you want to save a game? (y/n): ";
            std::string choice;
            std::cin >> choice;

            if(choice == "y"){
                game_state->save("game.txt");
            }

            try {
                return  enemy_field->ability_attack_cell(*user_abilities,
*enemy_ships);
            } catch (const std::exception& e) {
                std::cout << e.what() << std::endl;
```

```
                return false;
            }
        }

        bool Game::is_game_over(SeaBattlePlayground& field) {
            for(int i = 0; i < field.length; i++){
                for(int k = 0; k < field.width; k++){
                    if(field.field[k][i].cell_state == UNDAMAGED_SHIP ||
field.field[k][i].cell_state == DAMAGED_SHIP)
                        return false;
                }
            }
            return true;
        }

        Game::Game(SeaBattlePlayground* user_f, SeaBattlePlayground* enemy_f,
ShipManager*  user_s,  ShipManager*  enemy_s,  AbilityManager*  user_a,
GameState* game_s){
            round = 1;
            user_field = user_f;
            enemy_field = enemy_f;
            user_ships = user_s;
            enemy_ships = enemy_s;
            user_abilities = user_a;
            game_state = game_s;
        }

        void Game::start() {
            std::cout << "Do you want to load your game save? (y/n): ";
            std::string choice;
            std::cin >> choice;

            bool game_loaded = false;

            if(choice == "y"){
                game_loaded = true;
                game_state->load("game.txt");
            }
```

```
        while (true) {
            if(game_loaded){
                game_loaded = false;
            } else {
                if(round == 1) place_ships_randomly(user_field,
user_ships);
                place_ships_randomly(enemy_field, user_ships);
            }
            bool result = play_round();
            if(result == false) break;
        }
    }

    bool Game::play_round() {
        while (true) {
            std::cout << "Round " << round << " begins!" << std::endl;

            while (true) {
                user_turn();
                if (is_game_over(*enemy_field)) {
                    std::cout << "You win this round!" << std::endl;
                    ++round;
                    return true;
                }

                enemy_turn();
                if (is_game_over(*user_field)) {
                    std::cout << "You lose the game." << std::endl;
                    ++round;
                    return false;
                }
            }
        }
    }
```

Название файла: game.h

```
#ifndef GAME_H
#define GAME_H
```

```
#include <vector>
#include <memory>
#include "sea_battle_playground.h"
#include "ship_manager.h"
#include "ability_manager.h"
#include "ship.h"
#include "exceptions.h"
#include "game_state.h"

class Game {
private:
    SeaBattlePlayground* user_field;
    SeaBattlePlayground* enemy_field;
    ShipManager* user_ships;
    ShipManager* enemy_ships;
    AbilityManager* user_abilities;
    GameState* game_state;
    int round;

    void initialize_fields();
    void place_ships_randomly(SeaBattlePlayground* field,
ShipManager* ships);
    bool enemy_turn();
    bool user_turn();
    bool is_game_over(SeaBattlePlayground& field);
public:
    Game(SeaBattlePlayground* user_field, SeaBattlePlayground*
enemy_field, ShipManager* user_ships, ShipManager* enemy_ships,
AbilityManager* user_abilities, GameState* game_state);
    void load_save();
    void start();
    bool play_round();
};

#endif
```

Название файла: game_state.cpp

```
#include <vector>
#include <iostream>
```

```
#include <typeinfo>
#include "game_state.h"
#include "sea_battle_playground.h"
#include "ship_manager.h"
#include "ability_manager.h"

GameState::GameState(SeaBattlePlayground* player_f, SeaBattlePlayground* enemy_f, AbilityManager* player_ability_m, ShipManager* player_ship_m, ShipManager* enemy_ship_m){
    state_player_field = player_f;
    state_enemy_field = enemy_f;
    state_player_ability_manager = player_ability_m;
     state_player_ship_manager = player_ship_m;
     state_enemy_ship_manager = enemy_ship_m;
}

void GameState::save(const std::string& filename){
     std::ofstream ofs(filename, std::ios::binary);
     if (!ofs.is_open()) {
          throw std::runtime_error("Unable to open file for saving");
     }

     ofs << state_player_field->width << " " << state_player_field->length << "\n";
     save_player_field(ofs);
     save_enemy_field(ofs);

     save_player_ships(ofs);
     save_enemy_ships(ofs);

     ofs << state_player_ability_manager->abilities.size() << "\n";
     for (auto ability : state_player_ability_manager->abilities) {
          ofs << typeid(*ability).name() << "\n";
     }
}

void GameState::load(const std::string& filename) {
     std::ifstream ifs(filename, std::ios::binary);
     if (!ifs.is_open()) {
```

```
                throw std::runtime_error("Unable to open file for
loading");
           }

           int field_width, field_length;
           ifs >> field_width >> field_length;

           state_player_field->clear_field();
           state_enemy_field->clear_field();

           load_player_field(ifs);
           load_enemy_field(ifs);

           load_player_ships(ifs);
           load_enemy_ships(ifs);

           for(int i = 0; i < state_player_field->length; i++){
                for(int k = 0; k < state_player_field->width; k++){
                     int                    ship_index                    =
state_player_field->field[k][i].ship_index;
                     if(ship_index != -1){

     state_player_field->tie_the_ship(&state_player_ship_manager->ships[
i], k, i);
                     }
                }
           }

           for(int i = 0; i < state_enemy_field->length; i++){
                for(int k = 0; k < state_enemy_field->width; k++){
                     if(state_enemy_field->field[k][i].ship_index != -1){
                          int                  ship_id                  =
state_enemy_field->field[k][i].ship_index;

    state_enemy_field->tie_the_ship(&state_enemy_ship_manager->ships[i],
k, i);
                     }
                }
           }
```

```
            load_player_abilities(ifs);
      }

      void GameState::save_player_field(std::ofstream& ofs) {
            for(int i = 0; i < state_player_field->width; i++){
                  for(int k = 0; k < state_player_field->length; k++){
                        ofs << state_player_field->field[i][k].cell_state <<
"   "   <<   state_player_field->field[i][k].ship_index   <<   "   "   <<
state_player_field->field[i][k].vertical_orientation << " ";
                  }
                  ofs << "\n";
            }
      }

      void GameState::save_enemy_field(std::ofstream& ofs) {
            for(int i = 0; i < state_enemy_field->width; i++){
                  for(int k = 0; k < state_enemy_field->length; k++){
                        ofs << state_enemy_field->field[i][k].cell_state <<
"   "   <<   state_enemy_field->field[i][k].ship_index   <<   "   "   <<
state_enemy_field->field[i][k].vertical_orientation << " ";
                  }
                  ofs << "\n";
            }
      }

      void GameState::load_player_field(std::ifstream& ifs){
            char cell_state;
            int ship_index;
            bool vertical_orientation;
            for(int i = 0; i < state_player_field->width; i++){
                  for(int k = 0; k < state_player_field->length; k++){
                        ifs    >>    cell_state    >>    ship_index    >>
vertical_orientation;
                        state_player_field->field[i][k].cell_state        =
cell_state;
                        state_player_field->field[i][k].ship_index        =
ship_index;
```

```
	state_player_field->field[i][k].vertical_orientation =
vertical_orientation;
		}
	}
}

void GameState::load_enemy_field(std::ifstream& ifs){
	for(int i = 0; i < state_enemy_field->width; i++){
		for(int k = 0; k < state_enemy_field->length; k++){
			char cell_state;
			int ship_index;
			bool vertical_orientation;
				ifs >> cell_state >> ship_index >>
vertical_orientation;
				state_enemy_field->field[i][k].cell_state =
cell_state;
				state_enemy_field->field[i][k].ship_index =
ship_index;

	state_enemy_field->field[i][k].vertical_orientation =
vertical_orientation;
		}
	}
}

void GameState::save_player_ships(std::ofstream& ofs){
	ofs << state_player_ship_manager->quantity << "\n";

	for(int i = 0; i < state_player_ship_manager->quantity; i++){
		ofs << state_player_ship_manager->ships[i].length << " "
<< state_player_ship_manager->ships[i].condition << " " <<
state_player_ship_manager->ships[i].id << " ";
		for(int k = 0; k <
state_player_ship_manager->ships[i].length; k++){
			ofs << state_player_ship_manager->ships[i].hp_bar[k]
<< " ";
		}
		ofs << "\n";
```

```
        }
    }

    void GameState::save_enemy_ships(std::ofstream& ofs){
        ofs << state_enemy_ship_manager->quantity << "\n";

        for(int i = 0; i < state_enemy_ship_manager->quantity; i++){
            ofs << state_enemy_ship_manager->ships[i].length << " " <<
state_enemy_ship_manager->ships[i].condition << " " <<
state_enemy_ship_manager->ships[i].id << " ";
            for(int k = 0; k < state_enemy_ship_manager->ships[i].length; k++){
                ofs << state_enemy_ship_manager->ships[i].hp_bar[k]
<< " ";
            }
            ofs << "\n";
        }
    }

    void GameState::load_player_ships(std::ifstream& ifs){
        int quantity;
        ifs >> quantity;

        for (int i = 0; i < quantity; ++i) {
            int length;
            std::string condition;
            int ship_id;
            ifs >> length >> condition >> ship_id;

            state_player_ship_manager->ships[i].condition = condition;

            for(int k = 0; k < state_player_ship_manager->ships[i].length; k++){
                ifs >> state_player_ship_manager->ships[i].hp_bar[k];
            }
        }
    }

    void GameState::load_enemy_ships(std::ifstream& ifs){
```

```
            int quantity;
            ifs >> quantity;

            for (int i = 0; i < quantity; ++i) {
                  int length;
                  std::string condition;
                  int ship_id;
                  ifs >> length >> condition >> ship_id;

                  state_enemy_ship_manager->ships[i].condition = condition;

                  for(int k = 0; k < state_enemy_ship_manager->ships[i].length; k++){
                        ifs >> state_enemy_ship_manager->ships[i].hp_bar[k];
                  }
            }
      }

      void GameState::load_player_abilities(std::ifstream& ifs) {
            int ability_count;
            ifs >> ability_count;

            state_player_ability_manager->delete_abilities();

            for (int i = 0; i < ability_count; ++i) {
                  std::string ability_name;
                  ifs >> ability_name;

                  if (ability_name == typeid(DoubleDamage).name()) {
                        state_player_ability_manager->add_double_damage();
                  } else if (ability_name == typeid(Scanner).name()) {
                        state_player_ability_manager->add_scanner();
                  } else if (ability_name == typeid(RandomBombardment).name()) {
                        state_player_ability_manager->add_randbomb();
                  }
            }
      }
```

Название файла: game_state.h

```
#ifndef GAME_STATE_H
#define GAME_STATE_H

#include <fstream>
#include "sea_battle_playground.h"
#include "ability_manager.h"
#include "ship_manager.h"

class GameState {
public:
    SeaBattlePlayground* state_player_field;
    SeaBattlePlayground* state_enemy_field;
    AbilityManager* state_player_ability_manager;
    ShipManager* state_player_ship_manager;
    ShipManager* state_enemy_ship_manager;

    GameState(SeaBattlePlayground* player_field,
SeaBattlePlayground* enemy_field, AbilityManager* player_ability_manager,
ShipManager* player_ship_manager, ShipManager* enemy_ship_manager);
     void save(const std::string& filename);
    void load(const std::string& filename);

    friend std::ostream& operator<<(std::ostream& os, GameState&
state) {
            os << "Player Field:\n";
            state.state_player_field->display_your_playing_field();

            os << "Enemy Field:\n";

state.state_enemy_field->display_playing_field_for_enemy();

            return os;
     }

private:
    void save_player_field(std::ofstream& ofs);
    void save_enemy_field(std::ofstream& ofs);
```

```
    void load_player_field(std::ifstream& ifs);
    void load_enemy_field(std::ifstream& ifs);
    void save_player_ships(std::ofstream& ofs);
    void save_enemy_ships(std::ofstream& ofs);
    void load_player_ships(std::ifstream& ifs);
    void load_enemy_ships(std::ifstream& ifs);
    void load_player_abilities(std::ifstream& ifs);
};

#endif
```

Название файла: Makefile

```
all : game
main.o : main.cpp
	g++ -c main.cpp
ship.o : ship.cpp
	g++ -c ship.cpp
field_cell.o : field_cell.cpp
	g++ -c field_cell.cpp
ship_manager.o : ship_manager.cpp
	g++ -c ship_manager.cpp
sea_battle_playground.o : sea_battle_playground.cpp
	g++ -c sea_battle_playground.cpp
exceptions.o : exceptions.cpp
	g++ -c exceptions.cpp
ability.o : ability.cpp
	g++ -c ability.cpp
double_damage.o : double_damage.cpp
	g++ -c double_damage.cpp
random_bombardment.o : random_bombardment.cpp
	g++ -c random_bombardment.cpp
scanner.o : scanner.cpp
	g++ -c scanner.cpp
ability_manager.o : ability_manager.cpp
	g++ -c ability_manager.cpp
game_state.o : game_state.cpp
	g++ -c game_state.cpp
game.o : game.cpp
	g++ -c game.cpp
```

```
game : main.o ship.o field_cell.o ship_manager.o sea_battle_playground.o exceptions.o ability.o double_damage.o random_bombardment.o scanner.o ability_manager.o game_state.o game.o
	g++ main.o ship.o field_cell.o ship_manager.o sea_battle_playground.o exceptions.o ability.o double_damage.o random_bombardment.o scanner.o ability_manager.o game_state.o game.o -o game
	rm *.o
```