МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Объектно-ориентированное программирование
по профилю: Системы поддержки принятия решений
направления профессиональной подготовки: 09.03.04 Программная инженерия, бакалавриат

Тема: «Разработка компьютерной игры “Морской бой-2”»

Студенты: Юрин Данила Андреевич
Кошкин Артем Сергеевич
Сабирова Алина Руслановна
Группа: ИКБО-15-18
Работа представлена к защите_____(дата) _____/Юрин Д.А./
_____/Кошкин А.С./
_____/Сабирова А.Р./
(подпись и ф.и.о. студентов)

Руководитель: Свищёв Андрей Владимирович, ассистент кафедры ИиППО
Работа допущена к защите_____(дата)_____/Свищёв А.В./
(подпись и ф.и.о. руководителя)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. МИРЭА. 2019г.

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ

по дисциплине: Объектно-ориентированное программирование
по профилю: Системы поддержки принятия решений
направления профессиональной подготовки: 09.03.04 Программная инженерия, бакалавр

Студенты: Юрин Данила Андреевич
Кошкин Артем Сергеевич
Сабирова Алина Руслановна
Группа: ИКБО-15-18
Срок представления к защите _____
Руководитель: Свищёв Андрей Владимирович, ассистент кафедры ИиППО
Тема: «Разработка компьютерной игры “Морской бой-2”»
Исходные данные: IDE Visual Studio Community 2017
Перечень вопросов к разработке (основная задача), графические материалы (слайды):

1. Определение функциональных назначений разрабатываемой системы, определение состава прототипа
2. Проектирование UML-диаграммы классов для всей системы целиком
3. Проектирование алгоритмических решений для прототипа
4. Реализация разработанных решений
5. Оформление пояснительной записки
6. Разработка презентации

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Заведующий кафедрой ИиППО: _____ /В.А. Мордвинов/, дата:_____

Задание на КР выдал_____ (ассистент кафедры ИиППО, Свищёв А.В.)

«_____»_____201__г.

Задание на КР получил_____ (Юрин Д.А., Кошкин А.С., Сабирова А.Р)

«_____»_____201__г.

УДК 004.42

Юрин Д.А., Кошкин А.С., Сабирова А.Р./ «Разработка компьютерной игры “Морской бой-2”»/ **Курсовая работа** по дисциплине «Объектно-ориентированное программирование» профиля «Системы поддержки принятия решений» направления профессиональной подготовки бакалавриата 09.03.04. «Программная инженерия» (2ой семестр) / руководитель ассистент А.В. Свищёв / кафедра ИиППО Института ИТ РТУ МИРЭА – с. 45, илл. 2, ист. 5, (в т.ч. 1 на англ. яз.).

Целью работы является разработка компьютерной игры “Морской бой-2”.

В рамках работы осуществлены анализ предметной области, проектирование и разработка компьютерной игры “Морской бой-2”.

Yurin D.A., Koshkin A.S., Sabirova A.R. / “Development of the computer game “Sea Battle-2”” / Course work in the discipline “Object-Oriented Programming” of the profile “Decision Support Systems” of the training area Bachelor degree 09.03.04. “Software Engineering” (2nd semester) / Head Assistant A.V. Svishchev / Department of the Institute of IT & IPPO IT RTU MIREA - p. 45, ill. 2, sources 5, (including 1 in English).

The aim of the work is to develop a computer game “Sea battle-2”.

As part of the work, the analysis of the subject area, the design and development of the computer game “Sea battle-2” were carried out.

ОГЛАВЛЕНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

- **ПО** – программное обеспечение
- **ПК** – персональный компьютер
- **ПЭВМ** – персональные электронные вычислительные машины
- **HDD** – жесткий диск
- **UML** – унифицированный язык моделирования
- **ОС** – операционная система
- **КТ** – компьютерные технологии

ВВЕДЕНИЕ

Сегодня мы живём в таком мире, в котором практически невозможно обойтись без использования компьютерных технологий. Использование КТ охватывает все возможные сферы жизни человека, в частности это относится и к играм.

Развитие компьютерных систем позволило расширить игровую индустрию, путем создания совершенно новых проектов, а также переносом классических игр с бумажных и другого рода носителей на ПК. В современном мире игры всегда находятся у людей под рукой, так как они есть на всех устройствах: персональных компьютерах, смартфонах, умных часах и даже на телевизорах. Этой реализации и посвящена тема данной курсовой работы.

Выполнение курсовой работы должно опираться на положения СМКО МИРЭА 7.5.1/04.И.05-16 «Инструкция по организации и проведению курсового проектирования» от 06.12.2016.

АКТУАЛЬНОСТЬ ВЫБРАННОЙ ТЕМЫ

Несомненно, старые способы реализации игр никуда не делись, однако перенос игр на персональные компьютеры поспособствовал созданию более удобной среды, не требующей каких-либо ресурсов, помимо самого ПК. Также ПК позволяет хранить на себе огромное количество игр, доступ к которым можно будет осуществить мгновенно без каких-либо ограничений, если говорить о перенесенной классике. Использование информационных технологий является самым оптимальным вариантом.

1. КОНКРЕТИЗАЦИЯ ЗАДАЧИ

Необходимо разработать такую программу, которая позволит играть в игру “Морской бой” с персонального компьютера. Она должна предоставлять возможность играть против ПК, обеспечив максимально быстрое построение поля и удобство игры. Поведение ПК должно быть основано на логических алгоритмах.

2. ГРАНИЦА ПРИМЕНИМОСТИ ПРОГРАММЫ

Данное программное обеспечение предлагается к использованию широкой аудитории пользователей и не имеет какого-либо профессионального предназначения.

3. МАСШТАБИРУЕМОСТЬ

Коммерческое развертывание данной программы не предполагается.

4. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной курсовой работе рассматривается реализация программного средства, которое позволяет поиграть в классическую бумажную игру на персональном компьютере. Подразумевается одиночная игра против логических алгоритмов.

5. АНАЛИЗ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

Анализ функциональных требований будет предоставлен в виде Технического задания.

5.1. ОПИСАНИЕ ТРЕБОВАНИЙ К РАЗРАБАТЫВАЕМОЙ ПРОГРАММНОЙ СИСТЕМЕ С ТОЧКИ ЗРЕНИЯ ПОЛЬЗОВАТЕЛЯ

Необходимо разработать программу для реализации компьютерной игры. Должен быть реализован игровой процесс с максимально простым игровым оформлением, понятном любому пользователя.

5.1.1. Наименование программы "Морской бой-2".

5.1.2. Назначение и область разработки

Программа предназначена для получения возможности игры в Морской бой. Программа доступна в виде консольного приложения.

5.2. ТРЕБОВАНИЯ К ПРОГРАММЕ

5.2.1. Требования к функциональным возможностям:

Программа должна обеспечивать возможность использования следующих функций:

- Создание игрового поля

- Заполнение его по усмотрению пользователя
- Игровой процесс против ПК
- Перезапуск игры по ее окончанию

Общий список требований к функциональным возможностям программного обеспечения представлен в методических указаниях преподавателей Московского технологического университета: Зориной Н.В., Зорина Л.Б., Соболева О.В [5].

5.2.2. Требования к надежности

Требования к обеспечению надежного функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

- организацией бесперебойного питания технических средств
- использованием лицензионного программного обеспечения
- регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»
- регулярным выполнением требований ГОСТ 51188-98.
- защитой информации.
- испытанием программных средств на наличие компьютерных вирусов
- соблюдением целостности программного кода (исходного вида)

5.3. УСЛОВИЯ ЭКСПЛУАТАЦИИ

5.3.1. Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

5.3.2. Требования к составу и параметрам технических средств

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

- процессор версии Intel Pentium, не менее;
- оперативную память объемом 1 Гигабайт, не менее;
- 40 Гигабайт на HDD, не менее;
- операционную систему Windows версии XP, не менее;

5.4. ТРЕБОВАНИЯ К ИНФОРМАЦИОННОЙ И ПРОГРАММНОЙ СОВМЕСТИМОСТИ

Требования к исходным кодам и языкам программирования

Программа написана на языке C++

Требования к защите информации и программ

Требования не предъявляются.

Специальные требования

Программа должна обеспечивать работу, а также сохранение в файл данных.

5.5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.5.1. Предварительный состав программной документации

Состав программной документации должен включать в себя:

- техническое задание
- программу и методики испытаний
- руководство пользователя.

5.6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

5.6.1. Экономические преимущества разработки

Ориентировочная экономическая эффективность не рассчитывается. Аналогия не проводится ввиду уникальности предъявляемых требований к разработке.

5.7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

5.7.1. Стадии разработки

Разработка должна быть проведена в три стадии:

- разработка технического задания
- рабочее проектирование
- внедрение

5.7.2. Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

- разработка программы
- разработка программной документации
- испытания программы

На стадии внедрения должен быть выполнен этап разработки подготовка и передача программы.

5.7.3. Содержание работ по этапам

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

- постановка задачи
- определение и уточнение требований к техническим средствам
- определение требований к программе
- определение стадий, этапов и сроков разработки программы и документации на неё
- согласование и утверждение технического задания

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.
На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

- разработка, согласование и утверждение и методики испытаний
- проведение приемо-сдаточных испытаний
- корректировка программы и программной документации по результатам испытаний

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах Заказчика.

5.8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

5.8.1. Виды испытаний

Приемо-сдаточные испытания должны проводиться на объекте Заказчика в оговоренные сроки.

Приемо-сдаточные испытания программы должны проводиться согласно разработанной Исполнителем и согласованной Заказчиком Программы и методик испытаний.

Ход проведения приемо-сдаточных испытаний Заказчик и Исполнитель документируют в Протоколе проведения испытаний.

5.8.2. Общие требования к приемке работы

На основании Протокола проведения испытаний Исполнитель совместно с Заказчиком подписывает акт приемки-сдачи программы в эксплуатацию

6. РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ И АЛГОРИТМОВ

6.1. Разработка функциональных классов

Данная программа содержит шесть классов, один из которых отвечает за графический интерфейс. Класс <Player> является классом, отвечающим за действия игрока. Класс <aiplayer> является классом, отвечающим за игру ПК. Также есть классы <Board>, <cell>, <Ship>. Класс <Game> отвечает за графическую составляющую программы, а также за ход игрового процесса.

Графический интерфейс не предусмотрен, так как программа работает непосредственно в консоли.

Для принятия ключевых решений был использован комплект специальной литературы по языку C++ [1-4].

Полная реализация программы представлена в *ПРИЛОЖЕНИИ А.*

6.2. UML-диаграмма классов

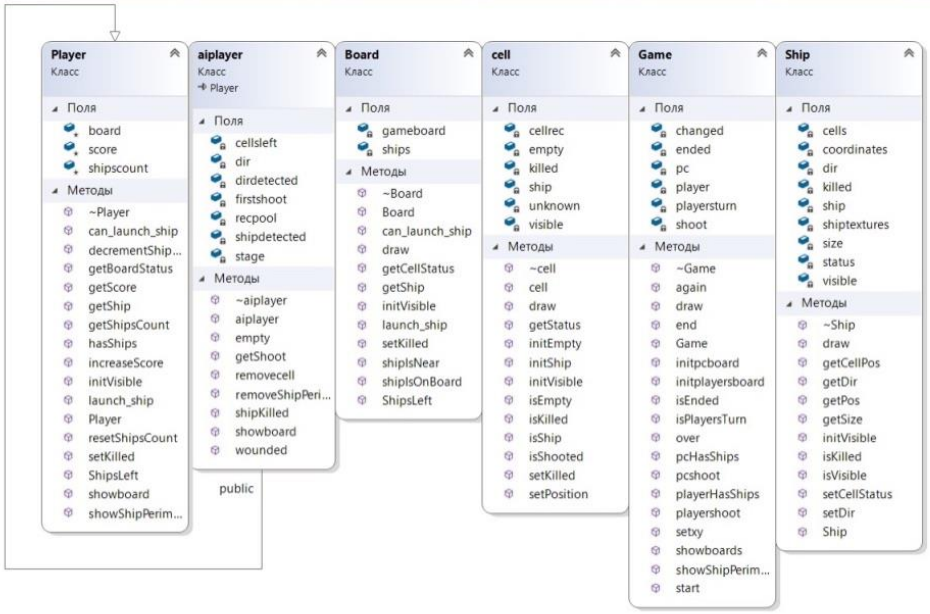

Рисунок 6.2.1 UML-диаграмма отношения обобщения классов

На рисунке 6.2.1 представлена диаграмма обобщения шести классов. Обобщение (оно же наследование) выражает наследование, в котором специализированный элемент (aiplayer) строится по спецификациям обобщенного элемента (Player). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя.

7. РЕАЛИЗАЦИЯ ПРОГРАММЫ

7.1. Описание работы программного средства

При запуске программы сначала необходимо расставить корабли на поле. Во время игры необходимо вводить номера клеток, в который будет совершен выстрел, в случае попадания по вражескому кораблю ход продолжится, если попадания не было, то ход передается ПК. ПК ведет

аналогично игроку. После завершения партии, расстановка кораблей сбрасывается, следовательно, с начала следующей партии необходимо повторить ввод.

7.2. Пример работы программы

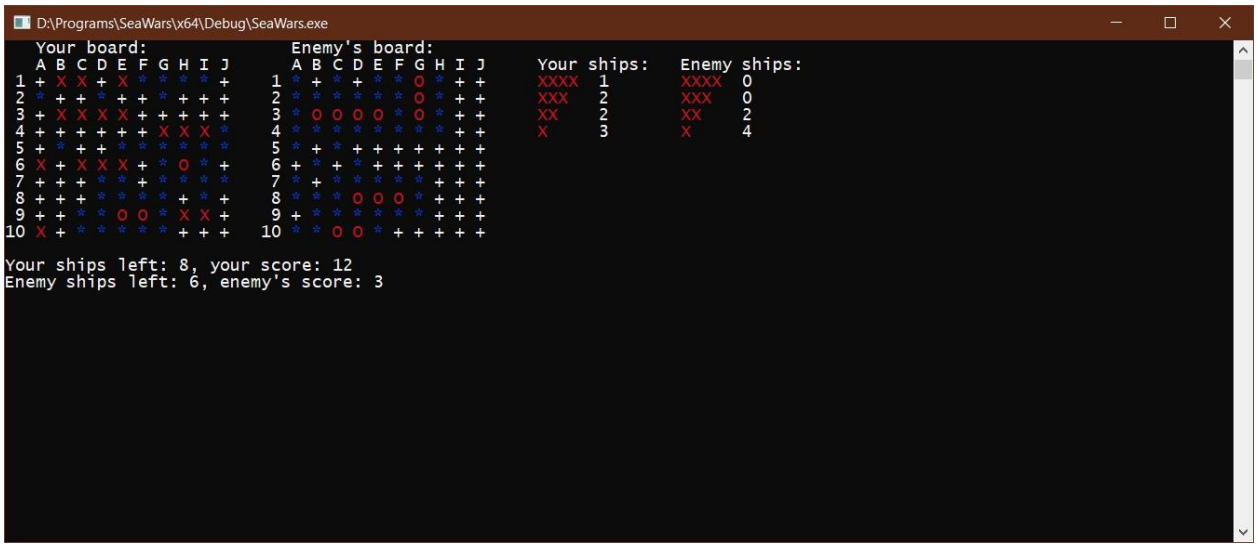

Рисунок 7.2.1 – Пример игрового процесса

8. ИНСТРУКЦИИ ПО ИСПОЛЬЗОВАНИЮ ПО

1. Для добавления корабля на поле необходимо ввести координаты поля, а также указать вертикальное, либо же горизонтальное расположение, а далее нажать ввод. Реализовать ввод в консоли.
2. Для того, чтобы совершить выстрел, нужно ввести координату клетки, в которую вы хотите попасть. Вводить адрес клетки нужно в консоли. В случае промаха ход перейдет ПК.
3. Во время хода противника следует дождаться его промаха, дальше право ходить вернется к вам.
4. Чтобы по окончанию игры, начать ее сначала, необходимо ввести “y” в консоль. Для того, чтобы покинуть игру необходимо ввести “n”.
5. Также, выйти из игры можно закрыв консоль.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы была создана игра «Морской Бой-2». Разработанная программа является приложением, позволяющим поиграть в классическую игру Морской бой на персональном компьютере. Программа позволяет легко составлять поле битвы, а также вести игру против логических алгоритмов, которые реализуются ПК.

Программа реализована максимально простым образом и предлагает управление через консоль, которое будет понятно любому пользователю. В ходе разработки приложения было необходимо изучение дополнительной литературы с целью максимально правильной разработки программы.

Разработка приложения позволила систематизировать все знания, накопленные за пройденный курс, и обобщила уже имеющиеся. Разработка программного средства помогла разобраться в основных тонкостях программирования на языке C++, а также понять принципы объектно-ориентированного стиля.

При выполнении курсовой работы и оформлении отчета по ней были использованы положения СМКО МИРЭА 7.5.1/04.И.05-16 «Инструкция по организации и проведению курсового проектирования» от 06.12.2016.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бьерн Страуструп. Язык программирования C++. Второе дополненное издание 1991 – 369 с. (дата обращения: 22.03.2019)
2. [Электронный ресурс] Visual Studio IDE. Microsoft. URL: https://visualstudio.microsoft.com/ru/ (дата обращения: 17.02.2019)
3. [Электронный ресурс] Морской бой. Правила игры. URL: https://ru.wikipedia.org/wiki/Морской_бой_(игра) (дата обращения: 20.02.2019)
4. [Электронный ресурс] Морской бой за 25 мс. Хабрахабр. URL: https://habr.com/ru/post/248061/ (дата обращения: 21.02.2019)
5. Зорина Н.В., Зорин Л.Б., Соболев О.В. Методические указания по выполнению курсовой работы для бакалавров. Часть I: методические материалы для студентов дневного отделения / под ред. А.А. Хлебникова – М.: МИРЭА, 2017 – 42 с. (дата обращения: 20.02.2019)

ПРИЛОЖЕНИЕ А. Листинг программы на языке C++

Seawars.cpp

```
#include "pch.h"
#include <iostream>
#include <iomanip>
#include <ctime>
#include "Game.h"
#include "Player.h"

using namespace std;

int main() {
	srand((int)time(NULL));

	while (1) {
		Game game;

		game.start();
		game.initplayersboard();
		game.initpcboard();
		bool gameover = false;
		while (!gameover) {
			gameover = game.over();
			while(1) {
				if (!game.playershoot()) {
					break;
				}
				if (game.over()) {
				gameover = true;
				break;
				}
			}
			if(gameover) {
				break;
			}

			while(1) {
				if (!game.pcshoot()) {
					break;
				}
				if (game.over()) {
				gameover = true;
				break;
				}
			}
			if(gameover) {
				break;
			}
		}

		game.end();

		if (!game.again()) {
			return 0;
		}
	}
}
```

Game.cpp

```cpp
#include "pch.h"
#include "Game.h"
#include <iostream>
#include <iomanip>

using namespace std;

Game::Game() {

}


Game::~Game() {

}

void Game::start() {
	player.resetShipsCount();
	pc.resetShipsCount();
	showboards();
}

void Game::end() {
	if (!pc.hasShips()) {
		cout << "You win!" << endl;
	}
	else {
		cout << "You lost!" << endl;
	}
}

bool Game::over() {
	if (!player.hasShips() || !pc.hasShips()) {
		return true;
	}
	else {
		return false;
	}
}

bool Game::pcHasShips() {
	if (pc.hasShips()) {
		return true;
	}
	else {
		return false;
	}
}

bool Game::playerHasShips() {
	if (player.hasShips()) {
		return true;
	}
	else {
		return false;
	}
}

void Game::initplayersboard() {

	cout << "If you wanna complete your board automatically, enter 'a'. Else enter
'm'" << endl;
	char prof;
	cin >> prof;
	if (prof != 'a' && prof != 'm') {
```

```
            cout << "You entered not right symbol, complete your board manually" <<
endl;
            prof = 'm';
      }

      for (int size = 4, count = 1; size >= 1, count <= 4; size--, count++) {
            for (int f = count; f >= 1; --f) {

                  if (prof == 'm') {

                        cout << "Let's launch your " << size << "-size ship" << endl;
                        if (size > 1) {
                              cout << "Enter position of the first cell (for example,
A2) of ship and direction (h or v):" << endl;
                        }
                        else {
                              cout << "Enter position of cell (for example, A2):" <<
endl;
                        }

                        char direct = 'v';
                        char x = 'a';
                        int y = 1;
                        cin >> x >> y;
                        if (x < 'A' || (x > 'J' && x < 'a') || x > 'j' || y < 1 || y >
10) {
                              cout << "Try again" << endl;
                              cin.clear();
                              ++f;
                        }
                        else {
                              Position pi;
                              pi.y = y - 1;
                              if (x >= 'a' && x <= 'j') {
                                    pi.x = x - 'a';
                              }
                              else if (x >= 'A' && x <= 'J') {
                                    pi.x = x - 'A';
                              }
                              if (size > 1) {
                                    cin >> direct;
                                    if (direct != 'h' && direct != 'H' && direct !=
'v' && direct != 'V') {
                                          cout << "You entered not right direction,
ship will be launched automatically in vertical position" << endl;
                                          direct = 'v';
                                    }
                              }
                              Ship ship(pi, size, direct);
                              if (player.can_launch_ship(pi.x, pi.y, ship.getDir(),
size)) {
                                    player.launch_ship(ship, true);
                              }
                              else {
                                    cout << "You can't launch ship here" << endl;
                                    f++;
                              }

                              showboards();
                        }
                  }
                  else {
                        char direct = 'v';
                        Position pi;
```

```
                        pi.y = rand() % 10;
                        pi.x = rand() % 10;

                        if (size > 1) {
                                direct = rand() % 2;
                                switch (direct) {
                                case 0:
                                        direct = 'v';
                                        break;
                                case 1:
                                        direct = 'h';
                                        break;
                                }
                        }
                        Ship ship(pi, size, direct);
                        if (player.can_launch_ship(pi.x, pi.y, ship.getDir(), size)) {
                                player.launch_ship(ship, true);
                        }
                        else {
                                f++;
                        }
                }
        }
        showboards();
}

void Game::initpcboard() {
        for (int size = 4, count = 1; size >= 1, count <= 4; size--, count++) {
                for (int f = count; f >= 1; --f) {

                        char direct = 'v';
                        Position pi;

                        pi.y = rand() % 10;
                        pi.x = rand() % 10;

                        if (size > 1) {
                                direct = rand() % 2;
                                switch (direct) {
                                case 0:
                                        direct = 'v';
                                        break;
                                case 1:
                                        direct = 'h';
                                        break;
                                }
                        }
                        Ship ship(pi, size, direct);
                        if (pc.can_launch_ship(pi.x, pi.y, ship.getDir(), size)) {
                                pc.launch_ship(ship, false);
                        }
                        else {
                                f++;
                        }
                }
        }
        showboards();
}

void print_line(Player &p, int i) {
        for (int g = 0; g < 10; g++) {
                if (p.getBoardStatus(i, g) == 'X' || p.getBoardStatus(i, g) == 'O') {
                        setcolor(RED, BLACK);
                        cout << setw(2) << p.getBoardStatus(i, g);
```

```
			}
			else if (p.getBoardStatus(i, g) == '*') {
				setcolor(BLUE, BLACK);
				cout << setw(2) << p.getBoardStatus(i, g);
			}
			else {
				setcolor(WHITE, BLACK);
				cout << setw(2) << p.getBoardStatus(i, g);
			}
			setcolor(WHITE, BLACK);
		}
}

void print_shipsleft(Player &p, int i) {
		if (i < 4) {
			setcolor(RED, BLACK);
			for (int g = 1; g <= 4 - i; g++) {
				cout << 'X';
			}
			setcolor(WHITE, BLACK);
			cout << setw(3+i) << p.ShipsLeft(4 - i);
		}
}

void Game::showboards() {
		system("CLS");
		setcolor(WHITE, BLACK);
		//COORD coord{ 0,0 };
		//COORD in{ 0,16 };
		//SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
		cout << setw(14) << "Your board:";
		cout << setw(14 + 14) << "Enemy's board:" << endl;
		cout << "  ";
		for (char i = 'A'; i <= 'A' + 9; i++) {
			setcolor(WHITE, BLACK);
			cout << setw(2) << i;
		}
		cout << "     ";
		for (char i = 'A'; i <= 'A' + 9; i++) {
			cout << setw(2) << i;
		}

		cout << setw(16) << "Your ships:";
		cout << setw(15) << "Enemy ships:";
		cout << endl;

		for (int i = 0; i < 10; i++) {
			cout << setw(2) << i + 1;
			print_line(player, i);

			cout << setw(5) << i + 1;
			print_line(pc, i);

			cout << "     ";
			print_shipsleft(player, i);

			cout << "       ";
			print_shipsleft(pc, i);

			cout << endl;
		}
		cout << endl;
		cout << "Your ships left: " << player.getShipsCount() << ", your score: " <<
player.getScore() << endl;
```

```
	cout << "Enemy ships left: " << pc.getShipsCount() << ", enemy's score: " <<
pc.getScore() << endl << endl;
	//SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), in);
}

void Game::showShipPerimeter(Ship ship, bool p) {

	Player *pl;
	if (p == true)
		pl = &player;
	else
		pl = &pc;

	pl->showShipPerimeter(ship);

}

bool Game::pcshoot() {

	Position pos = pc.getShoot();

	int x = pos.x;
	int y = pos.y;

	pc.removecell(x, y);

	if (player.getBoardStatus(y, x) == 'X') { //live ship cell
		pc.increaseScore();
		int i = player.setKilled(y, x);
		if (i != -1) {
			Ship s = player.getShip(i);
			showShipPerimeter(s, true);
			pc.shipKilled(s);
		}
		else {
			pc.wounded(y, x);
		}
		showboards();
		return true;
	}
	else if (player.getBoardStatus(y, x) == 'O') { //killed ship cell
		showboards();
		return true;
	}
	else if (player.getBoardStatus(y, x) == '+') { //empty cell
		player.initVisible(x, y);
		pc.empty(y, x);
		showboards();
		return false;
	}
	else {
		pc.empty(y, x);
	}
	//shoot again
	showboards();
	return true;
}

bool Game::playershoot() {
	char x = 'a';
	int y = 1;
	cin >> x >> y;

	/*x = 'a' + rand() % 10;
	y = 1 + rand() % 10;*/
```

```
	if (x < 'A' || (x > 'J' && x < 'a') || x > 'j' || y < 1 || y > 10) {
		cout << "Try again" << endl;
		cin.clear();
	}
	else {
		y = y - 1;
		if (x >= 'a' && x <= 'j') {
			x = x - 'a';
		}
		else if (x >= 'A' && x <= 'J') {
			x = x - 'A';
		}

		if (pc.getBoardStatus(y, x) == '+') {
			pc.initVisible(x, y);
			if (pc.getBoardStatus(y, x) == 'X') { //live ship cell
				player.increaseScore();
				int i = pc.setKilled(y, x);
				if (i != -1) {
					Ship s = pc.getShip(i);
					showShipPerimeter(s, false);
				}
				showboards();
				return true;
			}
			else if (pc.getBoardStatus(y, x) == 'O') { //killed ship cell
				showboards();
				return true;
			}
			else if(pc.getBoardStatus(y, x) == '*') { //empty cell
				showboards();
				return false;
			}
		}
		//shoot again
	}
	showboards();
	return true;
}

bool Game::again() {
	char answer = 0;
	do {
		cout << "Do U wanna play again?" << endl;
		cin >> answer;
	} while (answer != 'n' && answer != 'N' && answer != 'y' && answer != 'Y');
	if (answer == 'n' || answer == 'N') {
		return false;
	}
	else {
		return true;
	}
}
```

Game.h

```
#pragma once
#include "Player.h"
#include "aiplayer.h"
#include "ship.h"
#include "utils.h"

class Game {
private:
```

```
	Player player;
	aiplayer pc;
public:
	void start();
	void end();
	bool over();
	bool pcHasShips();
	bool playerHasShips();
	void initplayersboard();
	void initpcboard();
	void showShipPerimeter(Ship ship, bool p);
	void showboards();
	bool playershoot();
	bool pcshoot();
	bool again();
	Game();
	~Game();
};
```

Player.cpp

```
#include "pch.h"
#include "Player.h"


Player::Player() {
	shipscount = 10;
	score = 0;
}

Player::~Player() {

}

void Player::launch_ship(Ship &ship, bool visible) {
	board.launch_ship(ship, visible);
}

bool Player::can_launch_ship(int column, int row, direction direction, int size) {
	if(board.can_launch_ship(column, row, direction, size))
		return true;
	else
		return false;
}

char Player::getBoardStatus(int i, int g) {
	return board.getCellStatus(i, g);
}

bool Player::hasShips() {
	if (shipscount > 0) {
		return true;
	}
	else
		return false;
}

void Player::resetShipsCount() {
	shipscount = 10;
}

int Player::getShipsCount() {
	return shipscount;
}
```

```
void Player::decrementShipsCount() {
	shipscount--;
}

void Player::initVisible(int i, int g) {
	board.initVisible(i, g);
}

int Player::setKilled(int i, int g) {
	int f = board.setKilled(i, g);
	if (f != -1) {
		decrementShipsCount();
	}
	return f;
}

int Player::ShipsLeft(int size) {
	return board.ShipsLeft(size);
}

Ship Player::getShip(int i) {
	return board.getShip(i);
}

void Player::showShipPerimeter(Ship ship) {
	direction direction = ship.getDir();
	int row = ship.getPos().y;
	int column = ship.getPos().x;
	int size = ship.getSize();


	if (direction == h) {

		//vertical checking
		if (row - 1 < 0) {
			for (uint16_t j = 0; j < size; j++) {
				initVisible(column + j, row + 1);
			}
		}
		else if (row + 1 > 9) {
			for (uint16_t j = 0; j < size; j++) {
				initVisible(column + j, row - 1);
			}
		}
		else {
			for (uint16_t j = 0; j < size; j++) {
				initVisible(column + j, row - 1);
				initVisible(column + j, row + 1);
			}
		}

		//horizontal checking
		if (column - 1 < 0) {
			initVisible(column + size, row);
		}
		else if (column + size > 9) {
			initVisible(column - 1, row);
		}
		else {
			initVisible(column + size, row);
			initVisible(column - 1, row);
		}

		//diagonal checking
```

```
        //top left corner
        if (row - 1 < 0 && column - 1 < 0) {
                initVisible(column + size, row + 1);
        }
        //top right corner
        else if (row - 1 < 0 && column + size > 9) {
                initVisible(column - 1, row + 1);
        }
        //bottom left corner
        else if (row + 1 > 9 && column - 1 < 0) {
                initVisible(column + size, row - 1);
        }
        //bottom right corner
        else if (row + 1 > 9 && column + size > 9) {
                initVisible(column - 1, row - 1);
        }
        //left border of the field
        else if (column - 1 < 0 && row >= 0 && row <= 9) {
                initVisible(column + size, row - 1);
                initVisible(column + size, row + 1);
        }
        //right border of the field
        else if (column + size > 9 && row >= 0 && row <= 9) {
                initVisible(column - 1, row - 1);
                initVisible(column - 1, row + 1);
        }
        //top border of the field
        else if (row - 1 < 0 && column - 1 >= 0 && column + size <= 9) {
                initVisible(column - 1, row + 1);
                initVisible(column + size, row + 1);
        }
        //bottom border of the field
        else if (row + 1 > 9 && column - 1 >= 0 && column + size <= 9) {
                initVisible(column - 1, row - 1);
                initVisible(column + size, row - 1);
        }
        //not in the corner and not near the border of the field
        else {
                initVisible(column + size, row - 1);
                initVisible(column + size, row + 1);
                initVisible(column - 1, row + 1);
                initVisible(column - 1, row - 1);
        }
}
else {
        //vertical checking
        if (row - 1 < 0) {
                initVisible(column, row + size);
        }
        else if (row + size > 9) {
                initVisible(column, row - 1);
        }
        else {
                initVisible(column, row - 1);
                initVisible(column, row + size);
        }

        //horizontal checking
        if (column - 1 < 0) {
                for (uint16_t j = 0; j < size; j++) {
                        initVisible(column + 1, row + j);
                }
        }
        else if (column + 1 > 9) {
                for (uint16_t j = 0; j < size; j++) {
```

```
                        initVisible(column - 1, row + j);
                    }
                }
                else {
                    for (uint16_t j = 0; j < size; j++) {
                        initVisible(column + 1, row + j);
                        initVisible(column - 1, row + j);
                    }
                }

                //diagonal checking
                //top left corner
                if (row - 1 < 0 && column - 1 < 0) {
                    initVisible(column + 1, row + size);
                }
                //top right corner
                else if (row - 1 < 0 && column + 1 > 9) {
                    initVisible(column - 1, row + size);
                }
                //bottom left corner
                else if (row + size > 9 && column - 1 < 0) {
                    initVisible(column + 1, row - 1);
                }
                //bottom right corner
                else if (row + size > 9 && column + 1 > 9) {
                    initVisible(column - 1, row - 1);
                }
                //left border of the field
                else if (column - 1 < 0 && row - 1 >= 0 && row + size <= 9) {
                    initVisible(column + 1, row - 1);
                    initVisible(column + 1, row + size);
                }
                //right border of the field
                else if (column + 1 > 9 && row - 1 >= 0 && row + size <= 9) {
                    initVisible(column - 1, row - 1);
                    initVisible(column - 1, row + size);
                }
                //top border of the field
                else if (row - 1 < 0 && column - 1 >= 0 && column + 1 <= 9) {
                    initVisible(column - 1, row + size);
                    initVisible(column + 1, row + size);
                }
                //bottom border of the field
                else if (row + size > 9 && column - 1 >= 0 && column + 1 <= 9) {
                    initVisible(column - 1, row - 1);
                    initVisible(column + 1, row - 1);
                }
                //not in the corner
                else {
                    initVisible(column - 1, row - 1);
                    initVisible(column + 1, row - 1);
                    initVisible(column - 1, row + size);
                    initVisible(column + 1, row + size);
                }
        }
}

void Player::increaseScore() {
        score++;
}
int Player::getScore() {
        return score;
}
```

Player.h

```
#pragma once
#include "board.h"
#include "ship.h"

class Player {
protected:
	int shipscount;
	Board board;
	int score;
public:
	void launch_ship(Ship &ship, bool visible);
	bool can_launch_ship(int column, int row, direction direction, int size);
	char getBoardStatus(int i, int g);
	bool hasShips();
	void resetShipsCount();
	void decrementShipsCount();
	int getShipsCount();
	void initVisible(int i, int g);
	void showShipPerimeter(Ship ship);
	int setKilled(int i, int g);
	int ShipsLeft(int size);
	Ship getShip(int i);
	int getScore();
	void increaseScore();
	Player();
	~Player();
};
```

aiplayer.cpp

```
#include "pch.h"
#include "aiplayer.h"


aiplayer::aiplayer() {
	shipdetected = false;
	dirdetected = false;
	stage = unknown;
	for (int i = 0; i < 10; i++) {
		for (int g = 0; g < 10; g++) {
			Position p;
			p.y = i;
			p.x = g;
			cellsleft.push_back(p);
		}
	}
}

aiplayer::~aiplayer() {}

Position aiplayer::getShoot() {
	Position p;
	int s = cellsleft.size();
	p = cellsleft.at(rand() % s);
	recpool.shrink_to_fit();
	if (recpool.size() > 0 && shipdetected) {
		int s = (int)(recpool.size() - 1);
		p = recpool.at(s);
	}
	return p;
}
```

```
void aiplayer::wounded(int y, int x) {
	if (!shipdetected) {
		shipdetected = true;
		dirdetected = false;
		firstshoot.x = x;
		firstshoot.y = y;
		stage = one;

		Position p;
		if (y > 0) {
			p.x = x;
			p.y = y - 1;
			recpool.push_back(p);
		}
		if (y < 9) {
			p.x = x;
			p.y = y + 1;
			recpool.push_back(p);
		}
		if (x > 0) {
			p.x = x - 1;
			p.y = y;
			recpool.push_back(p);
		}
		if (x < 9) {
			p.x = x + 1;
			p.y = y;
			recpool.push_back(p);
		}
		return;
	}

	if (stage == one) {
		if (firstshoot.x != x) {
			dir = h;
		}
		else if (firstshoot.y != y) {
			dir = v;
		}
		recpool.clear();
		dirdetected = true;
		stage = two;
	}

	if (stage == two) {
			Position p;
			if (dir == h) {
				if (x < firstshoot.x) {
					if (x - 2 >= 0) {
						p.x = x - 2;
						p.y = y;
						recpool.push_back(p);
					}
					if (x - 1 >= 0) {
						p.x = x - 1;
						p.y = y;
						recpool.push_back(p);
					}
					if (firstshoot.x + 2 <= 9) {
						p.x = firstshoot.x + 2;
						p.y = y;
						recpool.push_back(p);
					}
					if (firstshoot.x + 1 <= 9) {
```

```
                        p.x = firstshoot.x + 1;
                        p.y = y;
                        recpool.push_back(p);
                }
        }
        else if (x > firstshoot.x) {
                if (firstshoot.x - 2 >= 0) {
                        p.x = firstshoot.x - 2;
                        p.y = y;
                        recpool.push_back(p);
                }
                if (firstshoot.x - 1 >= 0) {
                        p.x = firstshoot.x - 1;
                        p.y = y;
                        recpool.push_back(p);
                }
                if (x + 2 <= 9) {
                        p.x = x + 2;
                        p.y = y;
                        recpool.push_back(p);
                }
                if (x + 1 <= 9) {
                        p.x = x + 1;
                        p.y = y;
                        recpool.push_back(p);
                }
        }
}
else if (dir == v) {
        if (y < firstshoot.y) {
                if (y - 2 >= 0) {
                        p.y = y - 2;
                        p.x = x;
                        recpool.push_back(p);
                }
                if (y - 1 >= 0) {
                        p.y = y - 1;
                        p.x = x;
                        recpool.push_back(p);
                }
                if (firstshoot.y + 2 <= 9) {
                        p.y = firstshoot.y + 2;
                        p.x = x;
                        recpool.push_back(p);
                }
                if (firstshoot.y + 1 <= 9) {
                        p.y = firstshoot.y + 1;
                        p.x = x;
                        recpool.push_back(p);
                }
        }
        else if (y > firstshoot.y) {
                if (firstshoot.y - 2 >= 0) {
                        p.y = firstshoot.y - 2;
                        p.x = x;
                        recpool.push_back(p);
                }
                if (firstshoot.y - 1 >= 0) {
                        p.y = firstshoot.y - 1;
                        p.x = x;
                        recpool.push_back(p);
                }
                if (y + 2 <= 9) {
                        p.y = y + 2;
                        p.x = x;
```

```
                                        recpool.push_back(p);
                                }
                                if (y + 1 <= 9) {
                                        p.y = y + 1;
                                        p.x = x;
                                        recpool.push_back(p);
                                }
                        }
                }
                stage = three;
                return;
        }

    if (stage == three && recpool.size() > 0) {
        recpool.pop_back();
        return;
    }
}

void aiplayer::empty(int y, int x) {
    if (stage == one && recpool.size() > 0) {
        recpool.pop_back();
    }
    if (stage == three) {
        if (recpool.size() == 4 && recpool.at(3).x == x && recpool.at(3).y == y) {
            recpool.pop_back();
            recpool.pop_back();
        }
        else if ((recpool.size() == 2 || recpool.size() == 4) && recpool.at(1).x ==
x && recpool.at(1).y == y) {
            if ((recpool.at(0).x == x + 1) || (recpool.at(0).x == x - 1) ||
(recpool.at(0).y == y + 1) || (recpool.at(0).y == y - 1)) {
                if (recpool.size() == 2) {
                    recpool.pop_back();
                    recpool.pop_back();
                }
                else {
                    recpool.at(0) = recpool.at(2);
                    recpool.at(1) = recpool.at(3);
                    recpool.pop_back();
                    recpool.pop_back();
                }
            }
            else {
                recpool.pop_back();
            }
        }
        else if (recpool.size() > 0) {
            recpool.pop_back();
        }
    }
}

void aiplayer::removecell(int x, int y) {
    int g = -1;
    for (int i = 0; i < cellsleft.size(); i++) {
        if (cellsleft.at(i).x == x && cellsleft.at(i).y == y) {
            g = i;
            break;
        }
    }
    if (g != -1) {
        cellsleft.erase(cellsleft.begin() + g);
        cellsleft.shrink_to_fit();
    }
```

```
	cout << "Cells left: " << cellsleft.size() << endl;
}

void aiplayer::shipKilled(Ship ship) {
	recpool.clear();
	shipdetected = false;
	dirdetected = false;
	stage = unknown;
	removeShipPerimeter(ship);
}

void aiplayer::removeShipPerimeter(Ship ship) {
	direction direction = ship.getDir();
	int row = ship.getPos().y;
	int column = ship.getPos().x;
	int size = ship.getSize();


	if (direction == h) {

		//vertical checking
		if (row - 1 < 0) {
			for (uint16_t j = 0; j < size; j++) {
				removecell(column + j, row + 1);
			}
		}
		else if (row + 1 > 9) {
			for (uint16_t j = 0; j < size; j++) {
				removecell(column + j, row - 1);
			}
		}
		else {
			for (uint16_t j = 0; j < size; j++) {
				removecell(column + j, row - 1);
				removecell(column + j, row + 1);
			}
		}

		//horizontal checking
		if (column - 1 < 0) {
			removecell(column + size, row);
		}
		else if (column + size > 9) {
			removecell(column - 1, row);
		}
		else {
			removecell(column + size, row);
			removecell(column - 1, row);
		}

		//diagonal checking
		//top left corner
		if (row - 1 < 0 && column - 1 < 0) {
			removecell(column + size, row + 1);
		}
		//top right corner
		else if (row - 1 < 0 && column + size > 9) {
			removecell(column - 1, row + 1);
		}
		//bottom left corner
		else if (row + 1 > 9 && column - 1 < 0) {
			removecell(column + size, row - 1);
		}
		//bottom right corner
		else if (row + 1 > 9 && column + size > 9) {
```

```
                removecell(column - 1, row - 1);
        }
        //left border of the field
        else if (column - 1 < 0 && row >= 0 && row <= 9) {
                removecell(column + size, row - 1);
                removecell(column + size, row + 1);
        }
        //right border of the field
        else if (column + size > 9 && row >= 0 && row <= 9) {
                removecell(column - 1, row - 1);
                removecell(column - 1, row + 1);
        }
        //top border of the field
        else if (row - 1 < 0 && column - 1 >= 0 && column + size <= 9) {
                removecell(column - 1, row + 1);
                removecell(column + size, row + 1);
        }
        //bottom border of the field
        else if (row + 1 > 9 && column - 1 >= 0 && column + size <= 9) {
                removecell(column - 1, row - 1);
                removecell(column + size, row - 1);
        }
        //not in the corner and not near the border of the field
        else {
                removecell(column + size, row - 1);
                removecell(column + size, row + 1);
                removecell(column - 1, row + 1);
                removecell(column - 1, row - 1);
        }
}
else {
        //vertical checking
        if (row - 1 < 0) {
                removecell(column, row + size);
        }
        else if (row + size > 9) {
                removecell(column, row - 1);
        }
        else {
                removecell(column, row - 1);
                removecell(column, row + size);
        }

        //horizontal checking
        if (column - 1 < 0) {
                for (uint16_t j = 0; j < size; j++) {
                        removecell(column + 1, row + j);
                }
        }
        else if (column + 1 > 9) {
                for (uint16_t j = 0; j < size; j++) {
                        removecell(column - 1, row + j);
                }
        }
        else {
                for (uint16_t j = 0; j < size; j++) {
                        removecell(column + 1, row + j);
                        removecell(column - 1, row + j);
                }
        }

        //diagonal checking
        //top left corner
        if (row - 1 < 0 && column - 1 < 0) {
                removecell(column + 1, row + size);
```

```
		}
		//top right corner
		else if (row - 1 < 0 && column + 1 > 9) {
			removecell(column - 1, row + size);
		}
		//bottom left corner
		else if (row + size > 9 && column - 1 < 0) {
			removecell(column + 1, row - 1);
		}
		//bottom right corner
		else if (row + size > 9 && column + 1 > 9) {
			removecell(column - 1, row - 1);
		}
		//left border of the field
		else if (column - 1 < 0 && row - 1 >= 0 && row + size <= 9) {
			removecell(column + 1, row - 1);
			removecell(column + 1, row + size);
		}
		//right border of the field
		else if (column + 1 > 9 && row - 1 >= 0 && row + size <= 9) {
			removecell(column - 1, row - 1);
			removecell(column - 1, row + size);
		}
		//top border of the field
		else if (row - 1 < 0 && column - 1 >= 0 && column + 1 <= 9) {
			removecell(column - 1, row + size);
			removecell(column + 1, row + size);
		}
		//bottom border of the field
		else if (row + size > 9 && column - 1 >= 0 && column + 1 <= 9) {
			removecell(column - 1, row - 1);
			removecell(column + 1, row - 1);
		}
		//not in the corner
		else {
			removecell(column - 1, row - 1);
			removecell(column + 1, row - 1);
			removecell(column - 1, row + size);
			removecell(column + 1, row + size);
		}
	}
}
```

aiplayer.h

```
#pragma once
#include "Player.h"
enum stage {one, two, three, unknown};

class aiplayer : public Player {
private:
	vector<Position> recpool;
	Position firstshoot;
	bool shipdetected;
	bool dirdetected;
	direction dir;
	stage stage;
	vector<Position> cellsleft;
public:
	Position getShoot();
	void shipKilled(Ship ship);
	void wounded(int y, int x);
	void removecell(int x, int y);
	void removeShipPerimeter(Ship ship);
	void empty(int y, int x);
```

```
	aiplayer();
	~aiplayer();
};
```

board.cpp

```
#include "pch.h"
#include "board.h"


Board::Board() {
	vector<cell> rows;
	for (uint16_t row = 0; row < 10; row++) {
		for (uint16_t column = 0; column < 10; column++) {
			cell cell;
			cell.initEmpty();
			rows.push_back(cell);
		}
		gameboard.push_back(rows);
	}
}


Board::~Board() {

}

void Board::launch_ship(Ship ship, bool visible) {
	if (can_launch_ship(ship.getPos().x, ship.getPos().y, ship.getDir(),
ship.getSize())) {

		if (ship.getDir() == h) {
			for (uint16_t j = 0, ship_size = ship.getSize(); j < ship_size; j++)
{
				gameboard.at(ship.getPos().y).at(ship.getPos().x +
j).initShip();
				if (visible) {
					gameboard.at(ship.getPos().y).at(ship.getPos().x +
j).initVisible();
				}
			}
		}
		else {
			for (uint16_t j = 0, ship_size = ship.getSize(); j < ship_size; j++)
{
				gameboard.at(ship.getPos().y +
j).at(ship.getPos().x).initShip();
				if (visible) {
					gameboard.at(ship.getPos().y +
j).at(ship.getPos().x).initVisible();
				}
			}
		}
		Ship s = ship;
		ships.push_back(s);
	}
}

bool Board::can_launch_ship(int column, int row, direction direction, int size)
{
	if (shipIsOnBoard(column, row, direction, size) && !shipIsNear(column, row,
direction, size)) {
		return true;
	}
```

```
	return false;
}

bool Board::shipIsOnBoard(int column, int row, direction direction, int size) {
	if (direction == h) {
		for (uint16_t j = 0; j < size; j++) {
			if ((column + size) > 10)
				return false;
			else if (!gameboard.at(row).at(column + j).isEmpty())
				return false;
		}
	}
	else {
		for (uint16_t j = 0; j < size; j++) {
			if ((row + size) > 10)
				return false;
			else if (!gameboard.at(row + j).at(column).isEmpty())
				return false;
		}
	}

	return true;
}

bool Board::shipIsNear(int column, int row, direction direction, int size) {
	if (direction == h) {

		//vertical checking
		if (row - 1 < 0) {
			for (uint16_t j = 0; j < size; j++) {
				if (!gameboard.at(row + 1).at(column + j).isEmpty()) {
					return true;
				}
			}
		}
		else if (row + 1 > 9) {
			for (uint16_t j = 0; j < size; j++) {
				if (!gameboard.at(row - 1).at(column + j).isEmpty()) {
					return true;
				}
			}
		}
		else {
			for (uint16_t j = 0; j < size; j++) {
				if (!gameboard.at(row - 1).at(column + j).isEmpty() ||
!gameboard.at(row + 1).at(column + j).isEmpty()) {
					return true;
				}
			}
		}

		//horizontal checking
		if (column - 1 < 0) {
			if (!gameboard.at(row).at(column + size).isEmpty()) {
				return true;
			}
		}
		else if (column + size > 9) {
			if (!gameboard.at(row).at(column - 1).isEmpty()) {
				return true;
			}
		}
		else {
```

```
                        if (!gameboard.at(row).at(column + size).isEmpty() ||
!gameboard.at(row).at(column - 1).isEmpty()) {
                                return true;
                        }
                }

                //diagonal checking
                //top left corner
                if (row - 1 < 0 && column - 1 < 0) {
                        if (!gameboard.at(row + 1).at(column + size).isEmpty()) {
                                return true;
                        }
                }
                //top right corner
                else if (row - 1 < 0 && column + size > 9) {
                        if (!gameboard.at(row + 1).at(column - 1).isEmpty()) {
                                return true;
                        }
                }
                //bottom left corner
                else if (row + 1 > 9 && column - 1 < 0) {
                        if (!gameboard.at(row - 1).at(column + size).isEmpty()) {
                                return true;
                        }
                }
                //bottom right corner
                else if (row + 1 > 9 && column + size > 9) {
                        if (!gameboard.at(row - 1).at(column - 1).isEmpty()) {
                                return true;
                        }
                }
                //left border of the field
                else if (column - 1 < 0 && row >= 0 && row <= 9) {
                        if (!gameboard.at(row - 1).at(column + size).isEmpty() ||
!gameboard.at(row + 1).at(column + size).isEmpty()) {
                                return true;
                        }
                }
                //right border of the field
                else if (column + size > 9 && row >= 0 && row <= 9) {
                        if (!gameboard.at(row - 1).at(column - 1).isEmpty() ||
!gameboard.at(row + 1).at(column - 1).isEmpty()) {
                                return true;
                        }
                }
                //top border of the field
                else if ( row - 1 < 0 && column - 1 >= 0 && column + size <= 9) {
                        if (!gameboard.at(row + 1).at(column - 1).isEmpty() ||
!gameboard.at(row + 1).at(column + size).isEmpty()) {
                                return true;
                        }
                }
                //bottom border of the field
                else if (row + 1 > 9 && column - 1 >= 0 && column + size <= 9) {
                        if (!gameboard.at(row - 1).at(column - 1).isEmpty() ||
!gameboard.at(row - 1).at(column + size).isEmpty()) {
                                return true;
                        }
                }
                //not in the corner and not near the border of the field
                else {
                        if (!gameboard.at(row - 1).at(column + size).isEmpty() ||
                                !gameboard.at(row + 1).at(column + size).isEmpty() ||
                                !gameboard.at(row + 1).at(column - 1).isEmpty() ||
                                !gameboard.at(row - 1).at(column - 1).isEmpty()) {
```

```
                        return true;
                    }
                }
            }
            else {
                //vertical checking
                if (row - 1 < 0) {
                    if (!gameboard.at(row + size).at(column).isEmpty()) {
                        return true;
                    }
                }
                else if (row + size > 9) {
                    if (!gameboard.at(row - 1).at(column).isEmpty()) {
                        return true;
                    }
                }
                else {
                    if (!gameboard.at(row - 1).at(column).isEmpty() || !gameboard.at(row
+ size).at(column).isEmpty()) {
                        return true;
                    }
                }

                //horizontal checking
                if (column - 1 < 0) {
                    for (uint16_t j = 0; j < size; j++) {
                        if (!gameboard.at(row + j).at(column + 1).isEmpty()) {
                            return true;
                        }
                    }
                }
                else if (column + 1 > 9) {
                    for (uint16_t j = 0; j < size; j++) {
                        if (!gameboard.at(row + j).at(column - 1).isEmpty()) {
                            return true;
                        }
                    }
                }
                else {
                    for (uint16_t j = 0; j < size; j++) {
                        if (!gameboard.at(row + j).at(column + 1).isEmpty() ||
!gameboard.at(row + j).at(column - 1).isEmpty()) {
                            return true;
                        }
                    }
                }

                //diagonal checking
                //top left corner
                if (row - 1 < 0 && column - 1 < 0) {
                    if (!gameboard.at(row + size).at(column + 1).isEmpty()) {
                        return true;
                    }
                }
                //top right corner
                else if (row - 1 < 0 && column + 1 > 9) {
                    if (!gameboard.at(row + size).at(column - 1).isEmpty()) {
                        return true;
                    }
                }
                //bottom left corner
                else if (row + size > 9 && column - 1 < 0) {
                    if (!gameboard.at(row - 1).at(column + 1).isEmpty()) {
                        return true;
                    }
```

```
			}
			//bottom right corner
			else if (row + size > 9 && column + 1 > 9) {
				if (!gameboard.at(row - 1).at(column - 1).isEmpty()) {
					return true;
				}
			}
			//left border of the field
			else if (column - 1 < 0 && row - 1 >= 0 && row + size <= 9) {
				if (!gameboard.at(row - 1).at(column + 1).isEmpty() ||
!gameboard.at(row + size).at(column + 1).isEmpty()) {
					return true;
				}
			}
			//right border of the field
			else if (column + 1 > 9 && row - 1 >= 0 && row + size <= 9) {
				if (!gameboard.at(row - 1).at(column - 1).isEmpty() ||
!gameboard.at(row + size).at(column - 1).isEmpty()) {
					return true;
				}
			}
			//top border of the field
			else if (row - 1 < 0 && column - 1 >= 0 && column + 1 <= 9) {
				if (!gameboard.at(row + size).at(column - 1).isEmpty() ||
!gameboard.at(row + size).at(column + 1).isEmpty()) {
					return true;
				}
			}
			//bottom border of the field
			else if (row + size > 9 && column - 1 >= 0 && column + 1 <= 9) {
				if (!gameboard.at(row - 1).at(column - 1).isEmpty() ||
!gameboard.at(row - 1).at(column + 1).isEmpty()) {
					return true;
				}
			}
			//not in the corner
			else {
				if (!gameboard.at(row - 1).at(column - 1).isEmpty() ||
					!gameboard.at(row - 1).at(column + 1).isEmpty() ||
					!gameboard.at(row + size).at(column - 1).isEmpty() ||
					!gameboard.at(row + size).at(column + 1).isEmpty()) {
					return true;
				}
			}
		}
		return false;
}

char Board::getCellStatus(int i, int g) {
		return gameboard.at(i).at(g).getStatus();
}

void Board::initVisible(int i, int g) {
		gameboard.at(g).at(i).initVisible();
}

int Board::setKilled(int i, int g) {
		gameboard.at(i).at(g).setKilled();
		int f = 0;
		while (f < 10) {
			int z = 0;
			int s = ships.at(f).getSize();
			//looking for cell in ships array
			while (z < s) {
```

```
                        if (ships.at(f).getCellPos(z).y == i && ships.at(f).getCellPos(z).x
== g) { //found position
                                z++;
                                break;
                        }
                        z++;
                }
                z--;
                if (ships.at(f).getCellPos(z).y == i && ships.at(f).getCellPos(z).x == g) {
                        ships.at(f).setCellStatus(z); //set cell killed and check if the
whole ship is killed
                        break;
                }
                f++;
        }

        //ship killed
        if (ships.at(f).isKilled()) {
                return f;
        }

        //ship not killed
        return -1;
}

int Board::ShipsLeft(int size) {
        int left = 0;
        for (int i = 0; i < ships.size(); i++) {
                if (ships.at(i).getSize() == size && !ships.at(i).isKilled()) {
                        left++;
                }
        }
        return left;
}

Ship Board::getShip(int i) {
        return ships.at(i);
}
```

board.h

```
#pragma once
#include "cell.h"
#include "ship.h"
#include <vector>
#include <iostream>

using namespace std;

class Board {
private:
        vector<Ship> ships;
        vector< vector<cell> > gameboard;
public:
        void launch_ship(Ship ship, bool visible);
        bool can_launch_ship(int column, int row, direction direction, int size);
        bool shipIsOnBoard(int column, int row, direction direction, int size);
        bool shipIsNear(int column, int row, direction direction, int size);
        char getCellStatus(int i, int g);
        void initVisible(int i, int g);
        int ShipsLeft(int size);
        Ship getShip(int i);
        int setKilled(int i, int g);
        Board();
```

```
	~Board();
};
```

cell.cpp

```
#include "pch.h"
#include "cell.h"


cell::cell() {
	empty = false;
	unknown = true;
	killed = false;
	ship = false;
	visible = false;
}


cell::~cell() {
}

void cell::initEmpty() {
	empty = true;
	unknown = false;
	ship = false;
}

void cell::initShip() {
	ship = true;
	unknown = false;
	empty = false;
}

void cell::initVisible() {
	visible = true;
}

bool cell::isEmpty() {
	return empty;
}

bool cell::isShooted() {
	return !unknown;
}

bool cell::isShip() {
	return ship;
}

bool cell::isKilled() {
	return killed;
}

void cell::setKilled() {
	killed = true;
}

char cell::getStatus() {
	if (visible) {
		if (ship) {
			if (killed) {
				return 'O';
			}
			else {
```

```
                        return 'X';
                    }
                }
                else if (empty) {
                    return '*';
                }
        }
        return '+';
}
```

cell.h

```
#pragma once
class cell {
private:
        bool empty;
        bool unknown;
        bool ship;
        bool killed;
        bool visible;
public:
        void initEmpty();
        void initShip();
        void initVisible();
        bool isEmpty();
        bool isShip();
        bool isShooted();
        bool isKilled();
        void setKilled();
        char getStatus();
        cell();
        ~cell();
};
```

ship.cpp

```
#include "ship.h"


Ship::Ship(Position &pos, int length, char direct): coordinates(pos), size(length),
dir(setDir(direct)) {
        killed = false;
        for (unsigned i = 0; i < size; i++) {
                Position pi = pos;
                if (dir == h) {
                        pi.x = pos.x + i;
                        pi.y = pos.y;
                }
                else {
                        pi.x = pos.x;
                        pi.y = pos.y + i;
                }
                cells.push_back(pi);
                status.push_back(true);
        }
}

direction Ship::setDir(char direct) {
        if (direct == 'h' || direct == 'H') {
                return h;
        }
        else if (direct == 'v' || direct == 'V') {
                return v;
```

```
	}
	return v;
}

Position Ship::getPos() {
	return coordinates;
}

Position Ship::getCellPos(int i) {
	return cells.at(i);
}

void Ship::setCellStatus(int i) {
	status.at(i) = false;
	bool t = true;
	for (unsigned int g = 0; g < size; g++) {
		if (status.at(g) == true) {
			t = false;
		}
	}
	killed = t;
}

bool Ship::isKilled() {
	return killed;
}

direction Ship::getDir() {
	return dir;
}
int Ship::getSize() {
	return size;
}

Ship::~Ship() {

}
```

ship.h

```
#pragma once
#include "utils.h"
#include <vector>

using namespace std;
enum direction { h, v };

class Ship {
private:
	Position coordinates;
	unsigned size;
	direction dir;
	vector<Position> cells;
	vector<bool> status;
	bool killed;
public:
	Ship(Position &pos, int length, char direct);
	Position getPos();
	Position getCellPos(int i);
	direction getDir();
	void setCellStatus(int i);
	bool isKilled();
	int getSize();
	direction setDir(char direct);
```

```
	~Ship();
};
```

utils.cpp

```
#include "pch.h"
#include "utils.h"
#include <algorithm>


void setcolor(unsigned int color, unsigned int background_color) {
	HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(hCon, color | background_color * 16 + color);
}
```

utils.h

```
#pragma once
#include <ctime>
#include <cstdlib>
#include <map>
#include <windows.h>

#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define LIGHTGRAY 7
#define DARKGRAY 8
#define LIGHTBLUE 9
#define LIGHTGREEN 10
#define LIGHTCYAN 11
#define LIGHTRED 12
#define LIGHTMAGENTA 13
#define YELLOW 14
#define WHITE 15

struct Position {
	int x;
	int y;
	Position(int column = 0, int row = 0): x(column), y(row) {}
};

void setcolor(unsigned int color, unsigned int background_color);
```