Российская федерация

Ханты-Мансийский автономный округ – Югра

Департамент образования и науки

Сургутский государственный университет ХМАО

Политехнический институт

Кафедра автоматики и компьютерных систем

**Пояснительная записка**

К курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

Выполнил: Худолеев М. С.

студент группы 609-22

Проверил: Гришмановский П. В.

доцент кафедры автоматики и компьютерных систем

Сургут

2023 г.

Аннотация

Курсовой проект на тему: Морской бой в компьютерной реализации.

Выполнил – Худолеев М.С.

Руководитель – Гришмановский П.В.

Курсовой проект состоит из пояснительной записки и графической части.

Пояснительная записка включает разделы – введение, анализ, проектирование, разработка программного продукта, тестирование и заключение.

В разделе анализа приведено описание предметной области, обзор аналогов, сформулирована и обобщена частная задача, а также приведено техническое задание для программного продукта.

В разделе проектирования задача описана формально, разработана поведенческая модель программы, произведён аргументированный выбор типов данных и их количества для их представления в памяти, а также разработаны основные алгоритмы и приведены их блок-схемы.

В разделе разработки программного продукту приведены листинги исходного года разработанного программного продукта.

В разделе тестирования разработана модель организации тестирования, а также приведены проведённые тесты разработанного программного продукта.

Работа приведена на 43 листах с использованием 7 источников. Содержит 16 рисунков, 3 схемы и 15 листингов.

Задание

Морской бой - реализация игры.

Оглавление

[Аннотация 1](#_Toc138102223)

[Задание 2](#_Toc138102224)

[Введение 4](#_Toc138102225)

[Этап 1. Анализ 5](#_Toc138102226)

[1.1. Описание предметной области 5](#_Toc138102227)

[1.2. Обзор аналогов 6](#_Toc138102228)

[1.3. Обобщение частной задачи 9](#_Toc138102229)

[Этап 2. Проектирование 11](#_Toc138102230)

[2.1. Формальное описание задачи 11](#_Toc138102231)

[2.2. Поведенческая модель программы 11](#_Toc138102232)

[2.3. Представление данных в памяти при их обработке 12](#_Toc138102233)

[2.4. Основные алгоритмы 12](#_Toc138102234)

[Этап 3. Разработка программного продукта 15](#_Toc138102235)

[Этап 4. Тестирование 22](#_Toc138102236)

[4.1. Организация тестирования 22](#_Toc138102237)

[4.2. Тесты 22](#_Toc138102238)

[4.2.1. Тестирование на этапе расстановки кораблей 23](#_Toc138102239)

[4.2.2. Тестирование на основном этапе игры 26](#_Toc138102240)

[Заключение 29](#_Toc138102241)

[Использованная литература: 30](#_Toc138102242)

[Приложение 31](#_Toc138102243)

[Приложение A. Блок-схемы 31](#_Toc138102244)

[Приложение B. Листинги программного кода 31](#_Toc138102245)

Введение

Компьютерные игры стали неотъемлемой частью современной культуры, предоставляя людям возможность погрузиться в увлекательные виртуальные миры и испытать новый опыт. Разработка компьютерных игр является сложным и многогранным процессом, требующим не только креативности, но и глубоких знаний в области программирования.

Программирование, суть которого заключается в создании программных продуктов, играет ключевую роль в разработке компьютерных игр. Это искусство перевода идей и концепций в функционирующие виртуальные миры. При помощи программирования разработчики могут создавать уникальные игровые механики, визуальные эффекты, искусственный интеллект и многое другое.

Разработка компьютерных игр включает в себя несколько этапов. Начиная с создания концепции и дизайна игры, разработчики затем переходят к написанию кода, созданию графики, звукового оформления и игрового контента. Кроме того, важной частью процесса является тестирование игры на различных платформах, чтобы убедиться в ее стабильной работе и пользовательском удобстве.

Цель данного курсового проекта заключается в разработке компьютерной игры, основанной на популярной настольной игре "Морской бой", на языке программирования Си. В процессе создания игры будут использованы принципы и техники программирования, позволяющие реализовать интерактивный геймплей, отображение игрового поля, многопользовательский режим и другие функции, делающие оригинальную настольную игру захватывающей и увлекательной.

Этап 1. Анализ

1.1. Описание предметной области

Морской бой – это классическая настольная игра, в которой два игрока соревнуются друг с другом, пытаясь потопить корабли противника на специально размеченном поле. Игра имеет долгую историю и широкую популярность во всем мире.

Начнём с первого: морской бой – довольно молодая игра. Версии истории её появления разнятся, однако можно выделить средний временной промежуток появления её на свет – конец 19 – начало 20 века. По одной из версий игра была изобретена в 1970-х годах бурлаком Петром Кондратьевым, как способ отвлечься от тяжёлого труда (подробно рассмотрено в [1, 2]). С тех пор она получила широкое распространение, эволюционировала и успешно дошла до наших дней.

Обычные правила морского боя просты и понятны даже для новичков. Игровое поле обычно представляет собой сетку, где вертикальные столбцы обозначают буквы, а горизонтальные строки – цифры. К примеру, в одной из простейших вариации игры всё это расчерчивается на клеточной бумаге. Затем каждый игрок размещает одинаковый набор кораблей на своей половине поля, где суда не видны противнику. Корабли имеют разную длину и форму, от одноклеточных до нескольких клеток в длину (и даже до нескольких в ширину – в зависимости от правил конкретной версии игры). Цель игры – отгадать координаты кораблей противника и потопить их все, стреляя в выбранные клетки поля.

Классическая версия морского боя, как правило, состоит из двух игровых полей размера 10x10 клеток, обозначенных в виде сетки. Каждый игрок получает своё поле и размещает на нём свои корабли перед началом игры.

В классической версии используются следующие типы кораблей с заданными размерами: один корабль из четырех клеток (большой), два корабля из трех клеток (средние), три корабля из двух клеток (малые) и четыре корабля из одной клетки (однопалубные). Корабли могут быть размещены вертикально или горизонтально, но они не могут соприкасаться друг с другом ни углами, ни сторонами (эта и другие версии более подробно рассмотрены в [3]).

Геймплей довольно прост: игроки по очереди называют координаты на поле противника, пытаясь попасть в корабли противника. Например, игрок может сказать – "В-3" для атаки клетки B3 на поле противника. Если выстрел попадает в пустую клетку, то игрок сообщает "Мимо". Если выстрел попадает в корабль противника, игрок сообщает "Попал". Противник затем сообщает, был ли корабль потоплен, с помощью слова "Потоплен". Если атака была успешной (игрок попал по кораблю противника), то игрок снова делает ход, игнорируя очерёдность. Игроки отмечают результаты выстрелов на своих полях, чтобы отслеживать уже сделанные ходы. Побеждает тот игрок, который сумеет первым потопить все корабли противника.

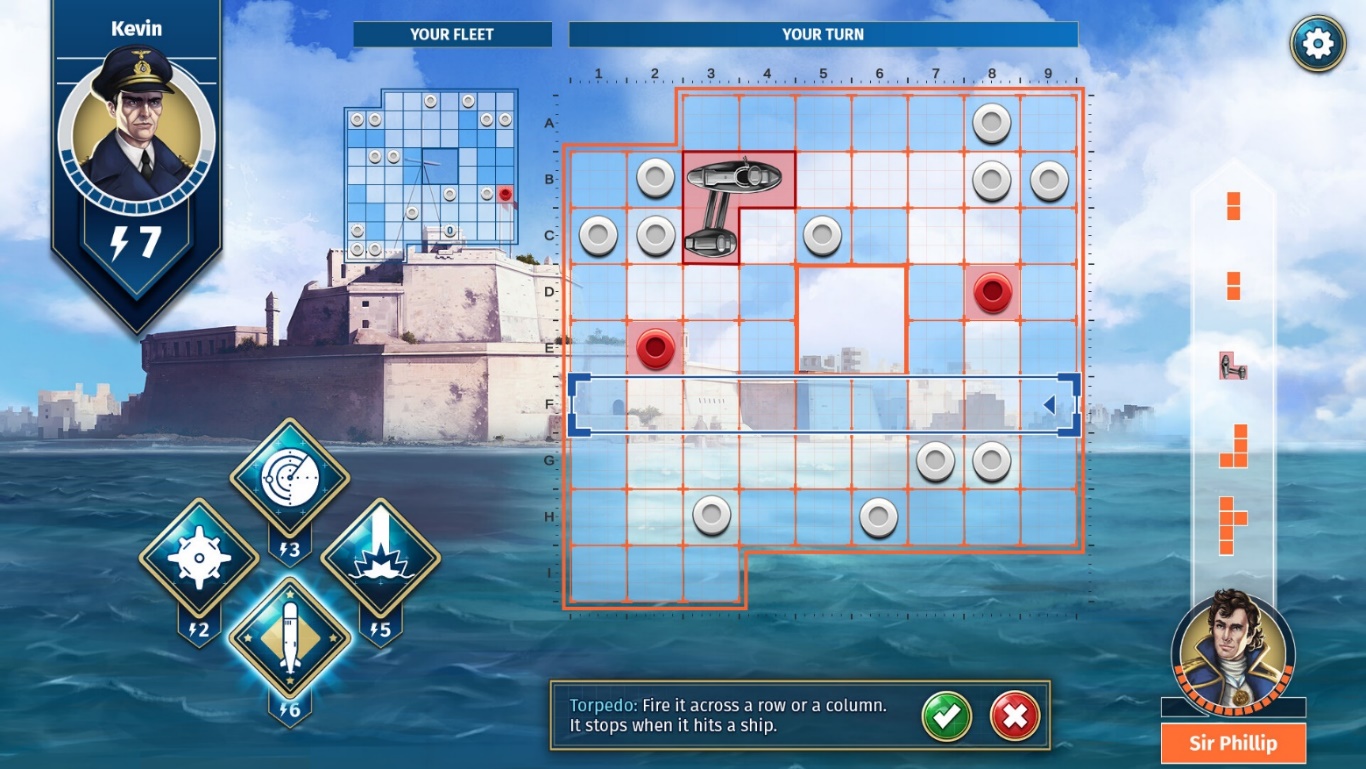
В современных версиях морского боя можно встретить различные вариации правил и дополнительные элементы. Например, некоторые версии добавляют возможность использования специальных умений или различных типов кораблей с разными характеристиками. Могут быть введены дополнительные элементы стратегии, такие как использование бонусов или специальных атак. Некоторые издания игры также предлагают разные тематические варианты, связанные с фильмами, книгами или компьютерными играми.

Морской бой также был успешно адаптирован для компьютерной игры. Такие версии оригинальной игры могут предлагать как одиночный геймплей против компьютерного противника, так и многопользовательский режим, где игроки могут соревноваться онлайн. Реализации морского боя также доступны на мобильных устройствах и позволяют играть в любое время и в любом месте. Рассмотрим эти версии поближе.

1.2. Обзор аналогов

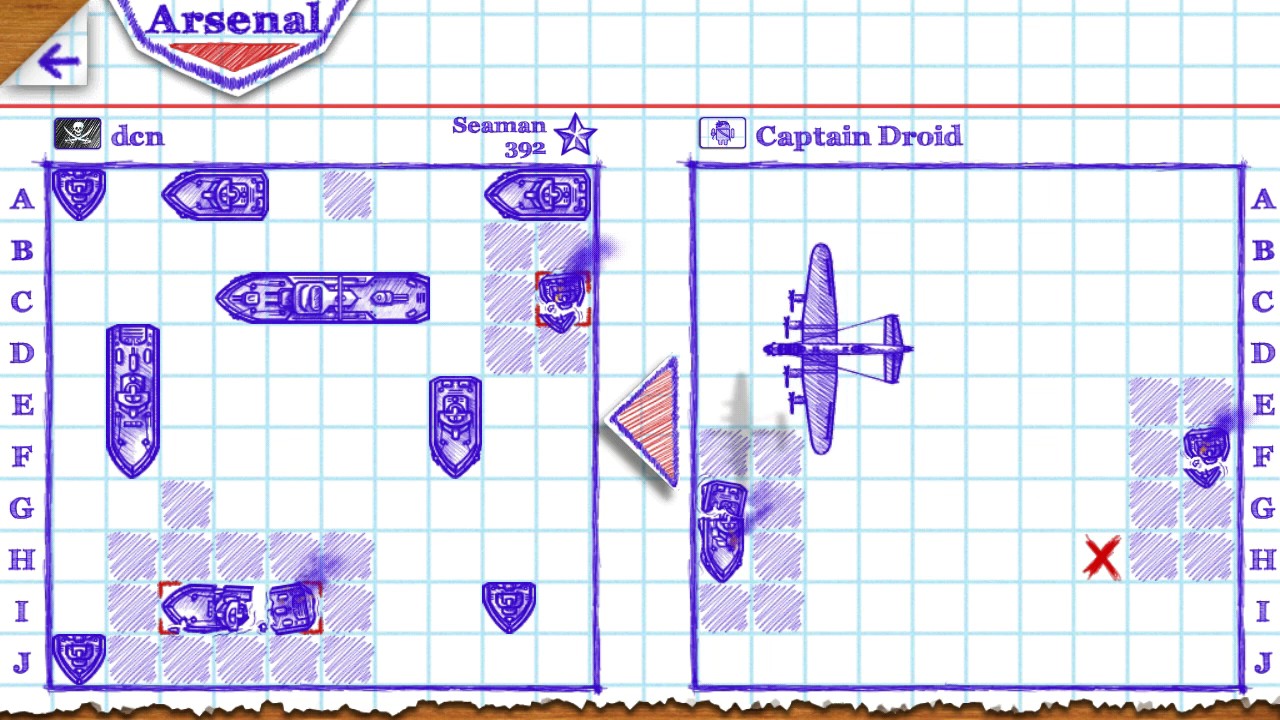
Вся представленная далее информация актуальна только на момент презентации этой работы.

1. Название проекта: «Hasbro’s BATTLESHIP» [4]

Рисунок 1. Геймплей первой игры-аналога

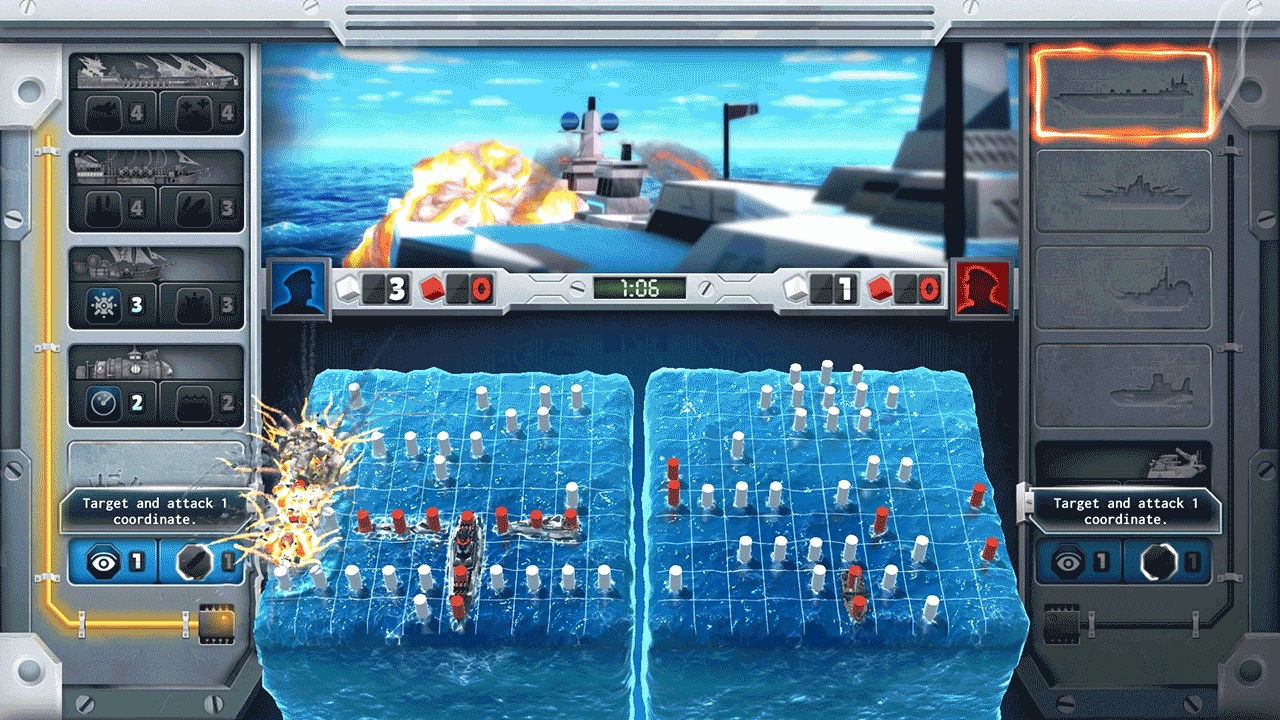
* Разработчик проекта: Marmalade Game Studio;
* Год выпуска: 2016;
* Статус проекта: по-прежнему доступен;
* Наличие онлайн-режима: да;
* Платформы: ПК (Windows);
* Онлайн-сервисы: Steam;
* Отзывы Steam: в основном положительные;
* Плюсы:
  + Обновленная графика и улучшенный визуальный опыт;
  + Возможность игры с компьютерным противником и онлайн-мультиплеером;
  + Различные режимы и настройки для более интересного игрового опыта.
* Минусы:
  + Могут возникать некоторые проблемы с соединением при игре в онлайн-режиме;
  + Некоторые игроки могут посчитать, что игра недостаточно инновационна и предлагает стандартный опыт игры в «Морской Бой».
* Общее представление: «Battleship» от Marmalade Game Studio предлагает классическую игру "Морской Бой" с обновленной графикой и возможностью играть как с компьютерным противником, так и с реальными соперниками онлайн. Игра имеет различные режимы и настройки, чтобы сделать игровой опыт более интересным и разнообразным. В том числе имеет различные размеры и формы игрового поля.

1. Название проекта: «Sea Battle 2» [5]

Рисунок 2. Геймплей второй игры-аналога

* Разработчик проекта: BYRIL;
* Год выпуска: 2014;
* Статус проекта: по-прежнему доступен;
* Наличие онлайн-режима: да;
* Платформы: мобильные устройства (iOS, Android);
* Онлайн-сервисы: App Store, Google Play;
* Рейтинг: Google Play – 4.7/5; App Store – 4.7/5;
* Плюсы:
  + Улучшенная графика и множество различных кораблей и специальных орудий;
  + Возможность игры с компьютерным противником и мультиплеером онлайн;
  + Гибкие настройки и возможность создания собственных правил игры.
* Минусы:
  + Некоторые игроки могут считать, что геймплей становится монотонным со временем;
  + Иногда могут возникать проблемы с соединением или задержкой в онлайн-режиме.
* Общее представление: «Sea Battle 2» от BYRIL предлагает современную интерпретацию классической игры «Морской Бой». Она имеет улучшенную графику, множество различных кораблей и специальных орудий. Игра позволяет играть с компьютерным противником или против реальных игроков в онлайн-режиме. Также есть возможность создавать свои собственные правила и настраивать игровой процесс.

1. Название проекта: «Battleship» [6]

Рисунок 3. Геймплей третьей игры-аналога

* Разработчик проекта: Ubisoft;
* Год выпуска: 2012;
* Статус проекта: по-прежнему доступен;
* Наличие онлайн-режима: да;
* Платформы: приставки (PlayStation 4, Xbox One, Wii);
* Онлайн-сервисы: PlayStation Network, Xbox Live, официальный онлайн-магазин Ubisoft;
* Рейтинг: Разнообразный рейтинг в зависимости от платформы;
* Плюсы:
  + Привлекательная графика и качественное визуальное оформление;
  + Возможность игры с компьютерным противником и мультиплеером онлайн;
  + Режим кампании с сюжетной линией, связанной с фильмом «Battleship».
* Минусы:
  + Игра может быть слишком простой и не предлагать достаточно сложных тактических решений;
  + Некоторые игроки могут считать, что игра ограничена в разнообразии и долгоиграемости.
* Общее представление: «Battleship» от Ubisoft – это адаптация игры «Морской Бой» в виде видеоигры. Игроки могут играть как с компьютерным противником, так и против других игроков в многопользовательском режиме онлайн. Игра предлагает различные режимы и настройки, а также имеет режим кампании с сюжетной линией, связанной с фильмом «Battleship».

1.3. Обобщение частной задачи

Написать прототип игры «Морской бой» для командной строки с возможностью добавления сетевых функций, таких как сетевая игра и таблица рекордов.

Форма поля и его размеры планируются быть константными. Само поле должно быть представлено массивом символов. Количество кораблей, их размеры и форма также планируются быть константными. Набор кораблей должен быть представлен массивом структур.

Задачи:

* Реализовать многопользовательский геймплей (формат PvP – человек против человека);
* Рассмотреть возможность добавления сетевых функций, а также реализовать их в случае, если такая возможность есть;
* Также в случае добавления сетевых функций добавить авторизацию пользователей, а также таблицу рекордов;
* Реализацию нужно выполнить под Windows без использования фреймворков;
* Реализовать игру, провести QA-тестирование.

Требования к программному продукту:

* Классический набор правил «Морского боя»;
* Игра двух пользователей на одном ПК;
* Консольный формат классического приложения Windows;
* Интерактивный пользовательский интерфейс с помощью текстового многоуровневого меню. Меню должно ознакомлять пользователя с игрой, её правилами, информацией о создателе.

Этап 2. Проектирование

В процессе проектирования выяснилось, что создание клиент-серверной архитектуры с сетевыми возможностями было довольно сложной и комплексной задачей, для решения которой были необходимы более обширные знания и умения, поэтому от этой идеи было решено отказаться.

2.1. Формальное описание задачи

В дополнение к пункту 1.3 и исходя из того, что правила будут соответствовать классическому морскому бою, можно выделить следующий набор вводных:

* Размер поля 10 на 10 клеток
* 1 большой корабль (1 на 4 клетки)
* 2 средних корабля (1 на 3 клетки)
* 3 малых корабля (1 на 2 клетки)
* 4 однопалубных корабля (1 на 1 клетку)

2.2. Поведенческая модель программы

Взаимодействие с пользователем предполагается производить через текстовое меню, открывающееся в командной строке при запуске программы. Игрокам нужно предоставить основную информацию о программе, игре, а также необходимо ознакомить их с правилами.

В начале игры расстановка кораблей игроками должна проводится по очереди –сначала первый игрок полностью расставляет корабли на своём поле, а затем второй – следующим образом:

1. Игрок вводит координату носа первого корабля (например: «E1»)
2. Игрок вводит направление расположения остальной части корабля относительно его носа (например: «down»)

При этом программа должна воспринимать как заглавные, так и строчные буквы при вводе.

После расстановки всех кораблей каждым игроком должен начинаться основной игровой цикл. Определение игрока, который будет проводить атаку первым необходимо сделать случайным, чтобы исключить постоянное преимущество у какого-либо пользователя.

Атака игроками будет проводится аналогично расстановке кораблей – пользователь вводит координату клетки для атаки (например: «D6»). О попадании в корабль или промахе мимо него будет сообщаться, а также будет сообщаться о потоплении корабля или победе игрока.

2.3. Представление данных в памяти при их обработке

Каждое поле должно быть представлено в виде двойного массива символьного типа 10 на 10. Координаты клетки для атаки или для установки носа корабля для передачи между функциями программы будут представлены в виде одной структуры, в которой будет две целочисленных переменных – координаты клетки по осям. Также набор кораблей будет представлен в виде массива структур, где каждый элемент массива будет представлять данные по одному из типов кораблей. Сама же структура должна иметь следующие данные: размер судна (целочисленный тип), количество судов этого типа (целочисленный тип) и его название (символьный тип, строка).

2.4. Основные алгоритмы

Рассмотрим основные алгоритмы, которые будут представлены в программе:



Схема 1. Алгоритм работы основного меню

Основное контекстное меню будет предлагать пользователю ознакомиться с правилами, информацией о разработчике, начать игру или же выйти из неё.



Схема 2. Общий алгоритм работы игровой функции

Здесь стоит отметить, что однотипные действия, по типу инициализации полей разбиты на отдельные функции в соответствии с общими принципами программирования KISS и DRY (подробно рассмотрено в [7]).

Алгоритм оптимизирован, чтобы иметь возможность начать игру заново без использования нерекомендуемых конструкций (по типу goto), а также повторять ходы до победы одного из игроков.

Возвращаемое функцией атаки значение в случае победы одного из игроков будет соответствовать номеру победившего игрока, в обратном же случае оно должно быть равно нулю. Также подразумевается, что игрок, который успешно провёл атаку, то есть попал по кораблю противника, должен снова делать ход (то есть атакующий не меняется) в случае удачной атаки (попадания по кораблю противника) игрок должен снова делать ход, без смены атакующего (описано в п.1.1). Поэтому возвращаемое значение функции атаки для такого случая должно быть отлично от нуля и отлично от номеров игроков (в стандартном случае 1 и 2), оно может быть равно, например, трём. В последнем условии if для переменной, значение которой равно возвращённому функцией как раз проверяется такой случай.

В приложении A.1. приведена блок-схема алгоритма атаки. Здесь под «клетка пуста» понимается отсутствие части корабля в ней. Также, потопленные части кораблей и промахи на поле будет обозначаться разными символами для того, чтобы:

1. Наглядно выводить поле для пользователя
2. Для правильной работы алгоритма распознавания потопленного судна, а также для правильного обозначения клеток вокруг потопленных судов как атакованных.

Также здесь клетки вокруг потопленных кораблей будут отмечаться как атакованные (но с промахом) с целью более понятного обозначения потопленных кораблей на поле и упрощения геймплея для игроков.

Для оптимизации проверка на конец игры будет проводиться только в случаях потопления судна, так как только при потоплении всех судов игра будет считаться завершённой, то есть вывод о завершении игры можно сделать только после уничтожения какого-то корабля.

Этап 3. Разработка программного продукта

В качестве языка программирования выбран C, так как это соответствует заданию, а качестве среды разработки – Microsoft Visual Studio 2022, как наиболее удобный для создания подобных программных продуктов.

Ниже приведены листинги программы

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <stdbool.h>
5. #include <string.h>

Листинг 1. Используемые библиотеки

Вызван заголовочный файл stdbool.h, так как в программе используется тип bool в качестве логической переменной. Аналогичные действия можно сделать и с типом int, но он весит 4 байта, а bool – 1 байт. Выбор сделан с целью сокращения затрат памяти.

1. #define FIELD\_SIZE 10
2. #define NUM\_SHIPS 4

Листинг 2. Используемые макросы

Макросы используются для упрощения работы с однотипными данными (размеры поля, количество типов судов). В случае изменения правил (например увеличения размеров поля), корректировка приложения не будет занимать много времени.

1. typedef char Field[FIELD\_SIZE][FIELD\_SIZE];
2. typedef struct {
3. int x;
4. int y;
5. } Coord;
6. typedef struct {
7. int size;
8. int num\_ships;
9. char\* name;
10. } Ship;

Листинг 3. Объявления новых имён для типов и групп типов данных (двух структур)

Критерии выбора соответствующих типов данных описаны в пункте 2.3.

1. void print\_field(Field field, bool show\_ships);
2. void init\_field(Field field);
3. int atack(Field defender\_field, Ship\* ships, int current\_player);
4. bool place\_ship(Field field, Ship\* ship, Coord coord, char direction);
5. bool is\_valid\_position(Field field, Ship\* ship, Coord coord, char direction);
6. Ship\* get\_ship\_by\_size(int size, Ship\* ships);
7. void print\_ships(Ship\* ships);
8. void init\_ships(Ship\* ships);
9. Coord get\_coord\_input(char\* prompt);
10. char get\_direction\_input(char\* prompt);
11. bool is\_coord\_on\_field(Coord coord);
12. Ship\* is\_ship\_sunk(Field field, Coord coord, Ship\* ships, int\* cells\_zero, int N);
13. bool is\_game\_over(Field field);
14. void main\_menu();
15. void game();
16. void about\_game();
17. void about\_dev();
18. void place\_ships(Field field, Ship\* ships, int player);

Листинг 4. Объявление всех функций, использующихся в программе

Функции объявлены в начале, так как описание всех функций для удобства сделано после описания главной функции (main)

1. int main() {
2. system("chcp 1251 > null");
3. main\_menu();
4. return 0;
5. }

Листинг 5. Описание главной функции

Здесь вызывается функция главного меню, так как все основные взаимодействия происходят через неё.

Описание функции основной страницы меню приведено в приложении B.1 (строки 49-87). Схема работы функции и её алгоритм приведены в пункте 2.4 (схема 1).

Описание функций страниц меню с информацией об игре и правилами, а также с информацией о разработчике приведено в приложении B.1. (строки 89-112). Функции отвечают за вывод на экран разделов интерактивного меню.

Описание основной игровой функции приведено в приложении B.1. (строки 114-193), схема её работы и алгоритм приведены в пункте 2.4 (схема 2).

1. void place\_ships(Field field, Ship\* ships, int player)
2. {
3. Coord coord;
4. int i = 0;
5. char direction = 0, c = 0;
6. for (i = 0; i < NUM\_SHIPS; i++) {
7. Ship\* ship = &ships[i];
8. direction = 0;
9. for (; ship->num\_ships > 0; ship->num\_ships--) {
10. printf("Player %d, place your ships:\n", player);
11. printf("Available ships:\n");
12. print\_ships(ships);
14. printf("Field:\n");
15. print\_field(field, true);
17. printf("Place your %s ship (amount: %d)\n", ship->name, ship->num\_ships);
19. coord = get\_coord\_input("Enter ship's nose coordinate (e.g. A1): ");
21. if (ship->size != 1)
22. direction = get\_direction\_input("Enter ship's direction (up, down, 70: left or right): ");
23. if (!place\_ship(field, ship, coord, direction)) {
24. ship->num\_ships++;
25. printf("Press [Enter] to continue\n");
26. while ((c = getchar()) != '\n' && c != EOF);
27. }
28. system("cls");
29. }
30. }
31. }

Листинг 6. Описание функции расстановки всех кораблей на поле игрока.

Данная функция позволяет расставить корабли на поле конкретного игрока, соответственно на вход в качестве формальных параметров она получает указатель на символьный массив типа игрового поля, указатель на первый элемент массива-набора доступных кораблей, а также номер игрока, расставляющего свои корабли.

Описание функции атаки приведено в приложении B.1. (строки 227-289). схема работы функции и её алгоритм приведены в пункте 2.4 (также см. блок-схему в приложении A.1.).

1. void print\_field(Field field, bool show\_ships) {
2. char\* L\_field = \*field;
4. printf(" ");
5. for (int i = 0; i < FIELD\_SIZE; i++) {
6. if (i <= 8)
7. printf("%c ", 'A' + i);
8. else
9. printf("%c ", 'A' + i);
10. }
11. printf("\n");
12. for (int i = 0; i < FIELD\_SIZE; i++) {
13. if (i <= 8)
14. printf("%d ", i + 1);
15. else
16. printf("%d ", i + 1);
17. for (int j = 0; j < FIELD\_SIZE; j++) {
18. if (show\_ships || \*L\_field == 'X' || \*L\_field == 'O')
19. printf("%c ", \*L\_field);
20. else
21. printf("~ ");
22. L\_field++;
23. }
24. printf("\n");
25. }
26. }

Листинг 7. Описание функции вывода игрового поля в консоль

Здесь стоит отметить, что имеется два режима вывода – скрытый для того, чтобы игроку отображалось поле противника без самих кораблей, и открытый для полного отображения своего поля для каждого игрока. Режим вывода передаётся как логическая переменная типа bool, где true означает открытый режим, а false – закрытый.

1. void init\_field(Field field) {
2. char\* L\_field = \*field;
3. for (int i = 0; i < FIELD\_SIZE; i++) {
4. for (int j = 0; j < FIELD\_SIZE; j++) {
5. \*L\_field = '~';
6. L\_field++;
7. }
8. }
9. }

Листинг 8. Описание функции инициализации игрового поля

В начале игры инициализирует массив символов типа игрового поля клетками с водой (символ «~»)

1. bool place\_ship(Field field, Ship\* ship, Coord coord, char direction) {
2. char\* L\_field = \*field;
3. if (is\_valid\_position(field, ship, coord, direction)) {
4. if (direction) {
5. for (int i = 0; i < ship->size; i++) {
6. \*(L\_field + coord.y \* FIELD\_SIZE + coord.x) = '\*';
8. if (direction == 'u') {
9. coord.y--;
10. }
11. else if (direction == 'd') {
12. coord.y++;
13. }
14. else if (direction == 'l') {
15. coord.x--;
16. }
17. else {
18. coord.x++;
19. }
20. }
21. }
22. else
23. \*(L\_field + coord.y \* FIELD\_SIZE + coord.x) = '\*';
24. return true;
25. }
26. else {
27. printf("Invalid placement data. Please check it and repeat input!\n");
28. return false;
29. }
30. }

Листинг 9. Описание функции расстановки судов

Функция позволяет установить конкретный корабль на поле игрока, вызывается функцией расстановки всех судов на игровом поле и на вход в качестве формальных переменных получает указатель на символьный массив типа игрового поля, указатель на структуру текущего типа устанавливаемого корабля, структуру-координату носа корабля, а также символьную переменную, отвечающую за направление остальной его части для установки. Функция вызывает другую для проверки валидности позиции для установки и, в случае валидной позиции, устанавливает туда корабль (в соответствующие позиции вносятся символы «\*», обозначающие действующую часть корабля.

Описание функции проверки валидности позиции для установки корабля приведено в приложении B.1. (строки 361-431). Функция также получает в качестве формальных параметров указатель на символьный массив типа игрового поля, указатель на структуру текущего типа устанавливаемого корабля, структуру-координату носа корабля, а также символьную переменную, отвечающую за направление остальной его части для установки. Функция проверяет все клетки, на которых планируется установить корабль на нахождение их на поле, а также наличие в них, или по соседству с ними других кораблей. Функция возвращает логическую переменную типа bool, обозначающую валидна позиция или нет.

1. Ship\* get\_ship\_by\_size(int size, Ship\* ships) {
2. for (int i = 0; i < NUM\_SHIPS; i++) {
3. if (ships[i].size == size) {
4. return &ships[i];
5. }
6. }
7. return NULL;
8. }

Листинг 10. Описание функции поиска корабля по размеру

1. void print\_ships(Ship\* ships) {
2. for (int i = 0; i < NUM\_SHIPS; i++) {
3. printf("%d %s[size: %d]\n", ships[i].num\_ships, ships[i].name, ships[i].size);
4. }
5. }

Листинг 11. Описание функции вывода доступных кораблей

1. void init\_ships(Ship\* ships) {
2. ships[0].size = 4;
3. ships[0].num\_ships = 1;
4. ships[0].name = "Battleship";
5. ships[1].size = 3;
6. ships[1].num\_ships = 2;
7. ships[1].name = "Cruiser";
8. ships[2].size = 2;
9. ships[2].num\_ships = 3;
10. ships[2].name = "Destroyer";
11. ships[3].size = 1;
12. ships[3].num\_ships = 4;
13. ships[3].name = "Submarine";
14. }

Листинг 12. Описание функции инициализации массива кораблей

Описание функции ввода и проверки клетки приведено в приложении B.1. (строки 466-530). При вводе проверяется количество вводимых символов – не более 3 и не менее 2, что соответствует следующим шаблонам: «A1» - «J1», «A10» - «J10». Затем посимвольно проверяется нахождение буквенной и численной координат на игровом поле. Функция возвращает полученную координату при корректном вводе, при некорректном предлагает проверить данные и попробовать ввод снова.

1. char get\_direction\_input(char\* prompt) {
2. while (true) {
3. char input[10];
4. printf("%s", prompt);
5. fflush(stdin);
6. fgets(input, sizeof(input), stdin);
7. fflush(stdin);
8. size\_t len = strlen(input);
9. if (len > 0 && input[len - 1] == '\n') {
10. input[len - 1] = '\0';
11. }
13. if (len >= 2) {
14. switch (input[0]) {
15. case 'u':
16. case 'U':
17. return 'u';
18. case 'd':
19. case 'D':
20. return 'd';
21. case 'l':
22. case 'L':
23. return 'l';
24. case 'r':
25. case 'R':
26. return 'r';
27. }
28. }
30. printf("Invalid direction. Please enter up, down, left or right.\n");
31. }
32. }

Листинг 13. Описание функции ввода направления установки судна на поле

Здесь присутствует следующая особенность реализации алгоритма – направление берётся и проверяется по первой букве написанного слова. Например, от «up» в качестве символа для проверки будет взят первый символ «u», и уже он будет сравниваться с шаблонами: «u», «r», «d» и «l» соответственно (буквы могут быть как строчные, так и заглавные). В общем случае, например при вводе корректного названия направления установки, такой способ проверки будет работать корректно, однако о такой особенности будет упомянуто в руководстве пользователя, и при дальнейшей работе над проектом эта особенность будет исправлена. Функция возвращает символьную переменную, отвечающую за направление.

1. bool is\_game\_over(Field field) {
2. char\* L\_field = \*field;
3. int i = 0;
4. for (i = 0; i < FIELD\_SIZE \* FIELD\_SIZE; i++) {
5. if (\*L\_field == '\*') {
6. return false;
7. }
8. L\_field++;
9. }
10. return true;
11. }

Листинг 14. Описание функции проверки окончания игры

Здесь производится прохождение по полю и проверка всех ячеек на наличие функционирующих (не потопленных) частей кораблей. Функция возвращает логическую переменную типа bool, обозначающую окончена игра или нет.

Описание функции проверки потопления судна приведено в приложении B.1. (строки 577-644). От заданной координаты осуществляется проход во все 4 стороны вокруг с подсчётом подбитых клеток (для возврата указателя на конкретную структуру – тип судна). Как только встречается живая клетка алгоритм прекращает работу и возвращает нулевой указатель.

1. bool is\_coord\_on\_field(Coord coord) {
2. return coord.x >= 0 && coord.x < FIELD\_SIZE&& coord.y >= 0 && coord.y < FIELD\_SIZE;
3. }

Листинг 15. Описание функции проверки координаты на нахождение на поле

Этап 4. Тестирование

4.1. Организация тестирования

Тестирование будет проходить следующим образом: на каждом этапе программы со вводом каких-либо данных будет введено несколько наборов корректных и некорректных данных.

4.2. Тесты

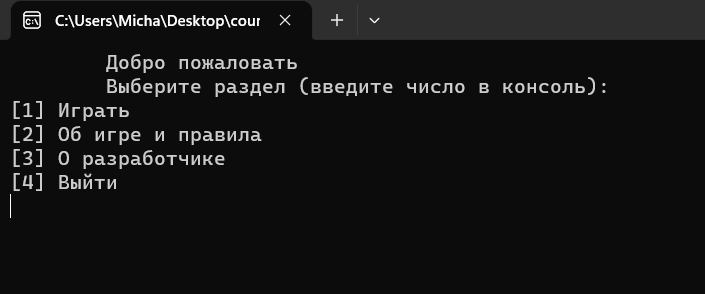


Рисунок 4. Основное меню

Проверим работу разделов меню, отвечающих за знакомство игроков с игрой и разработчиком:

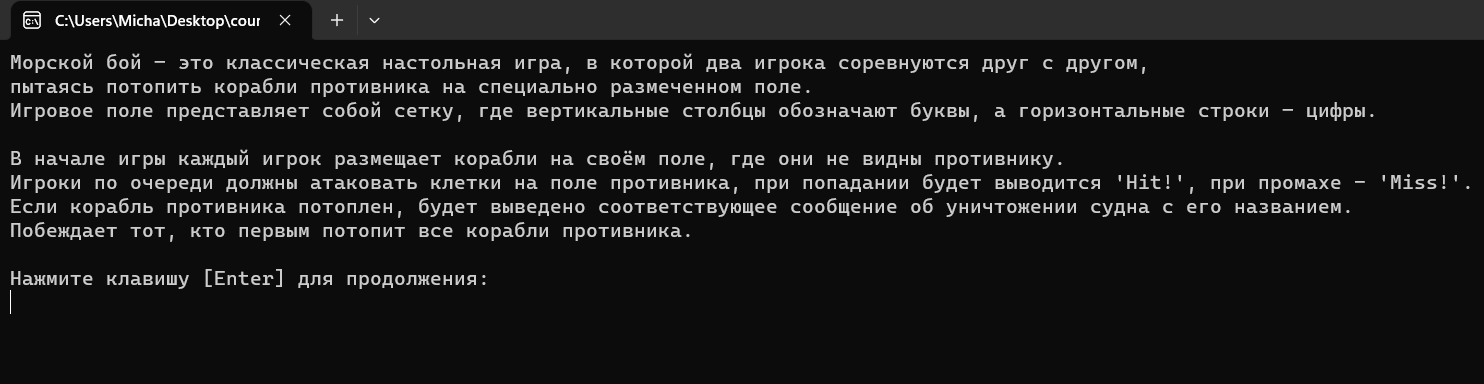


Рисунок 5. Страница меню с информацией об игре

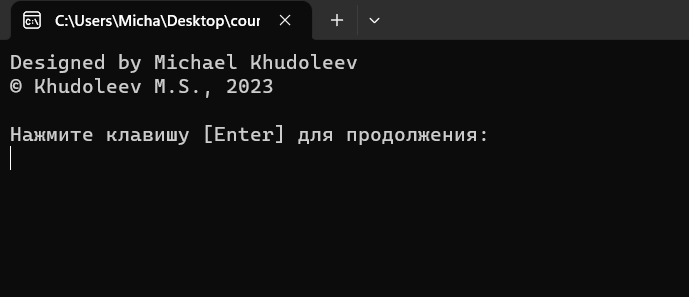


Рисунок 6. Страница меню с информацией о разработчике

Попробуем ввести некорректное значение:

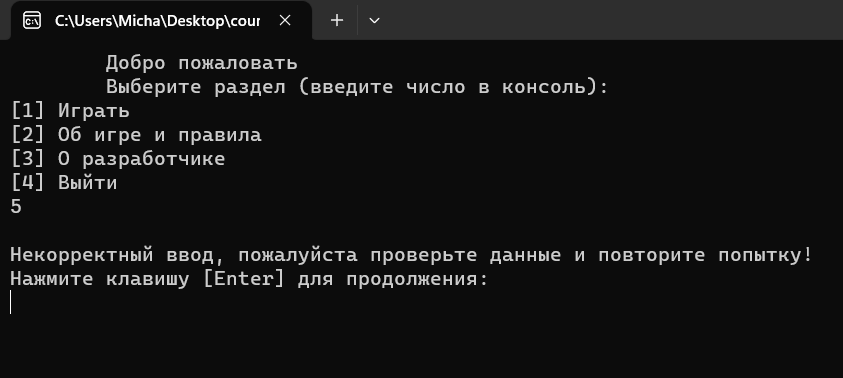


Рисунок 7. Ввод некорректных данных в главном меню

4.2.1. Тестирование на этапе расстановки кораблей

Перейдём к игре:

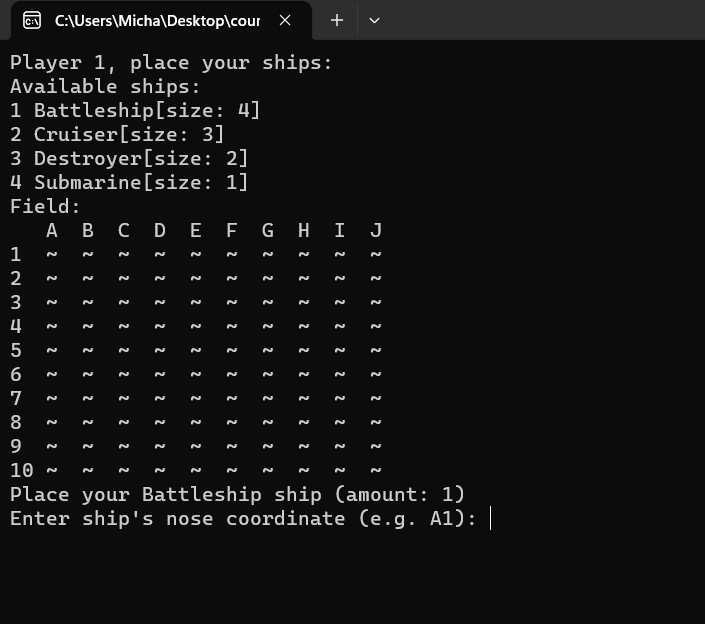


Рисунок 8. Интерфейс меню расстановки кораблей

Проведём корректную расстановку кораблей в уязвимых местах (стенки, углы поля):

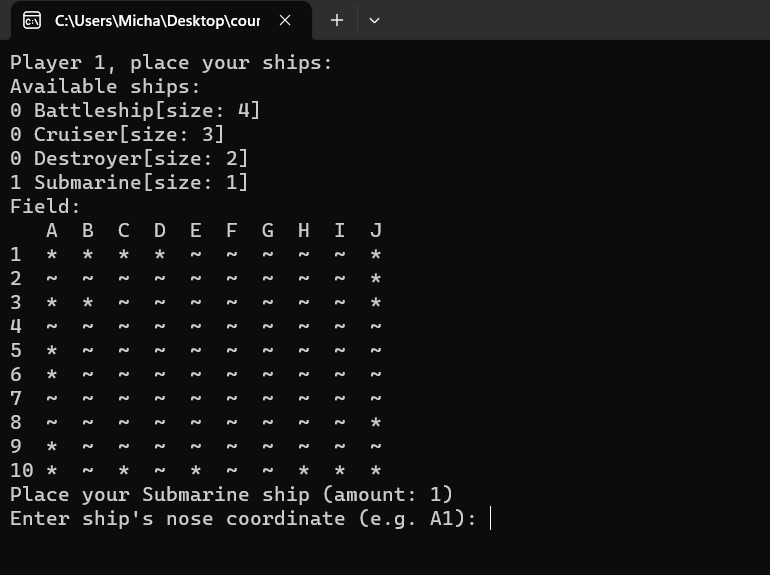


Рисунок 9. Корректная расстановка кораблей

Попробуем теперь задать некорректные данные:

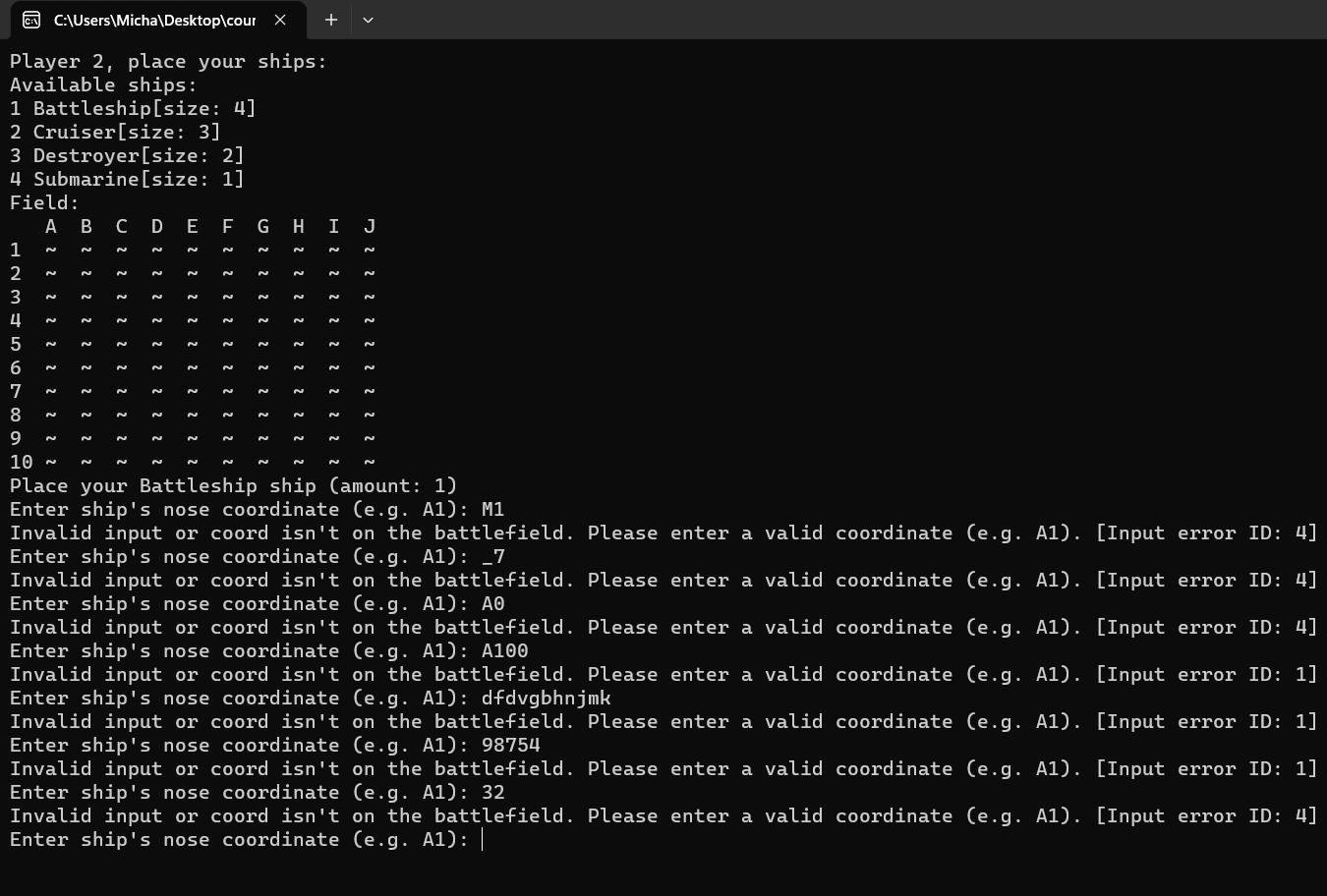


Рисунок 10. Попытка некорректного ввода

Здесь приведены различные типы тестовых данных: некорректный ввод буквенной, числовой координат, а также произвольный ввод.

Теперь протестируем некорректный ввод направления, например установка части корабля за полем, или произвольное направление:

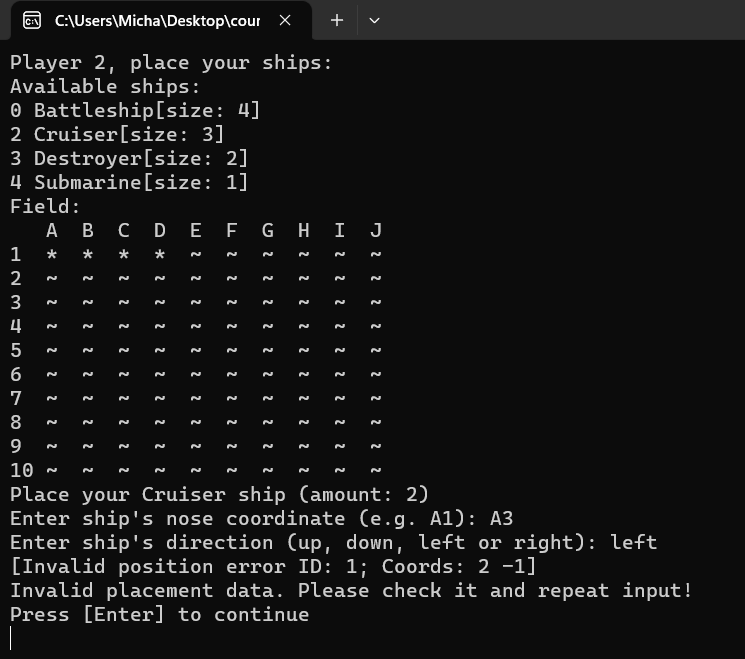


Рисунок 11. Попытка установить корабль за поле

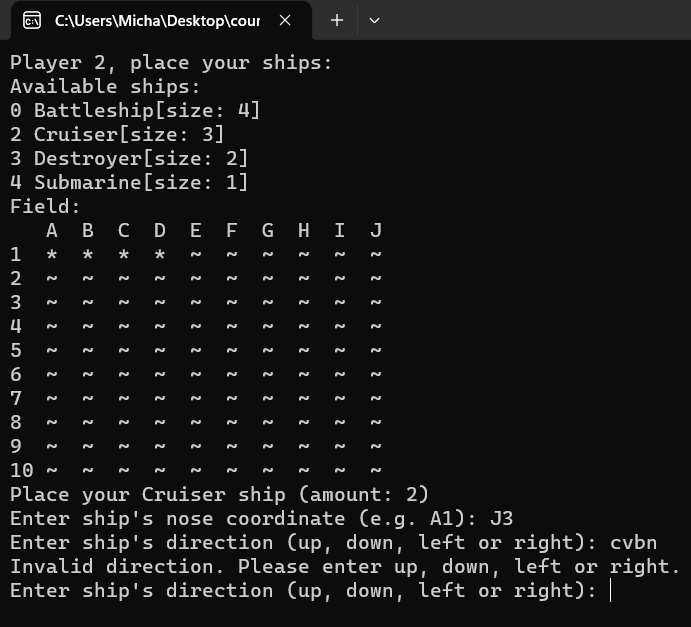


Рисунок 12. Попытка некорректного ввода направления

Здесь необходимо учитывать особенность работы функции получения направления установки корабля, описанную на этапе 3.

Также проверим возможность установки корабля вплотную к другому:

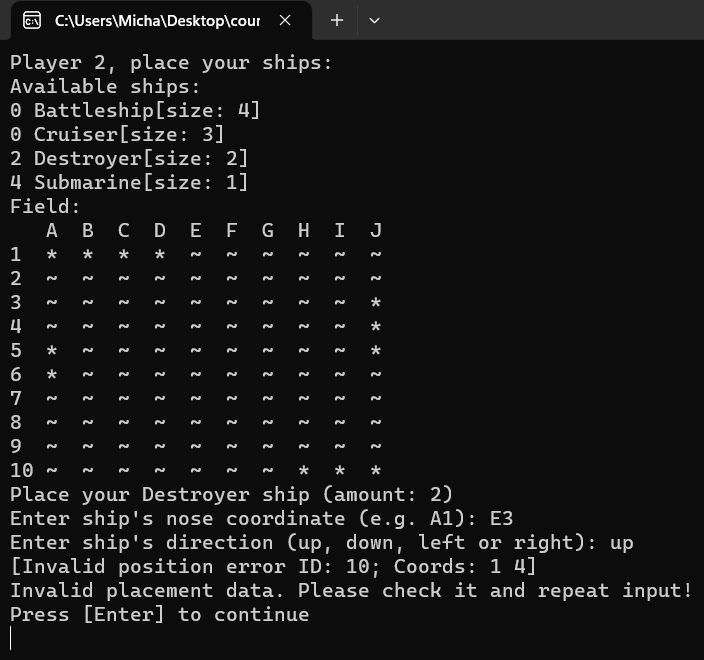


Рисунок 13. Попытка установки корабля вплотную к другому

Произведена неудачная попытка установить двухпалубный корабль вплотную к четырёхпалубному.

4.2.2. Тестирование на основном этапе игры

Проверки на этапе расстановки кораблей в целом работают корректно, перейдем к тестированию игры на основном этапе. Попробуем атаковать клетку за полем или в поле клетки для атаки ввести произвольный набор данных:

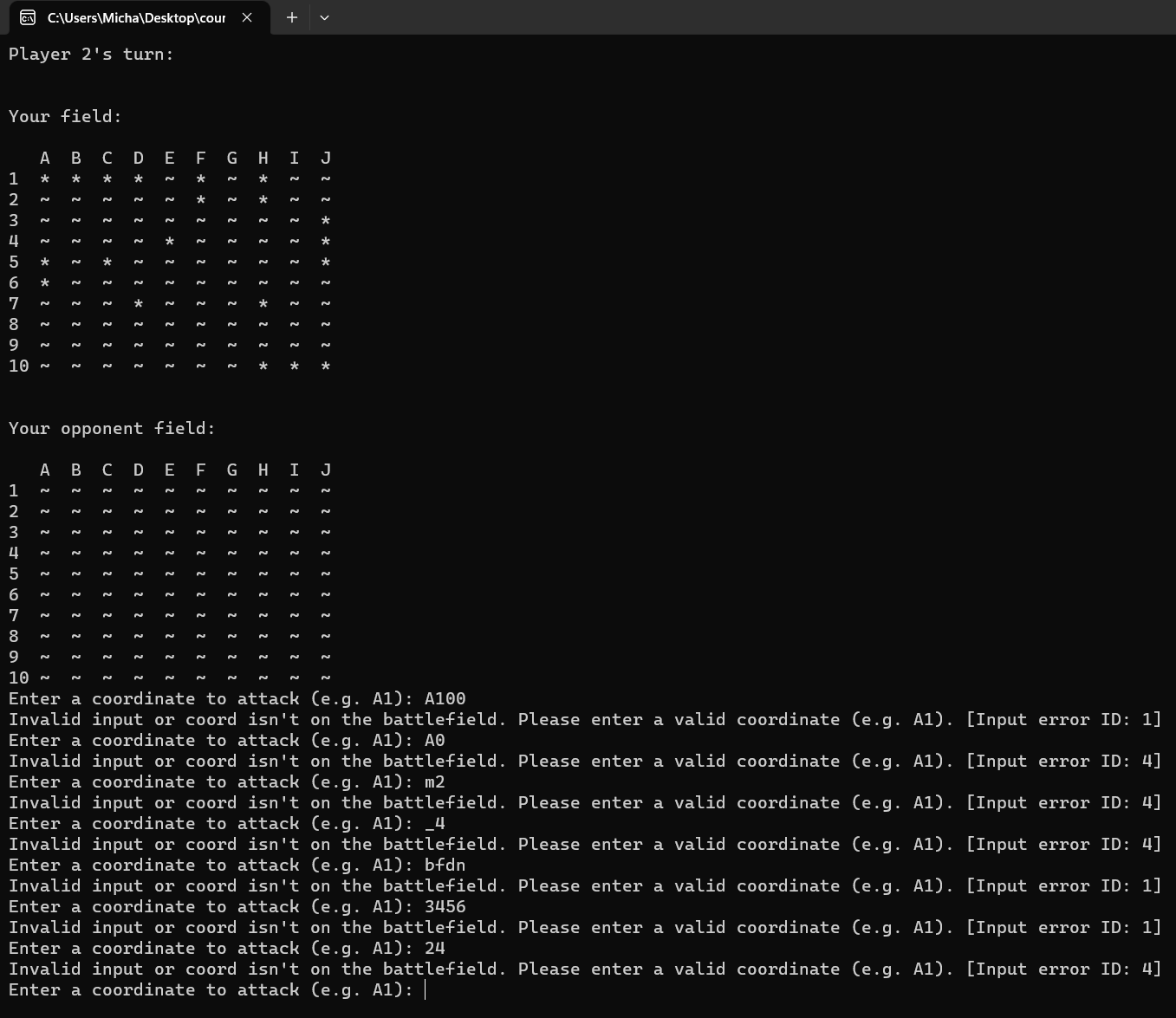


Рисунок 14. Попытка некорректного ввода

Здесь снова приведены различные типы тестовых данных: некорректный ввод буквенной, числовой координат, а также произвольный ввод.

Проверки работают корректно, проверим работу алгоритма определения потопления судна:

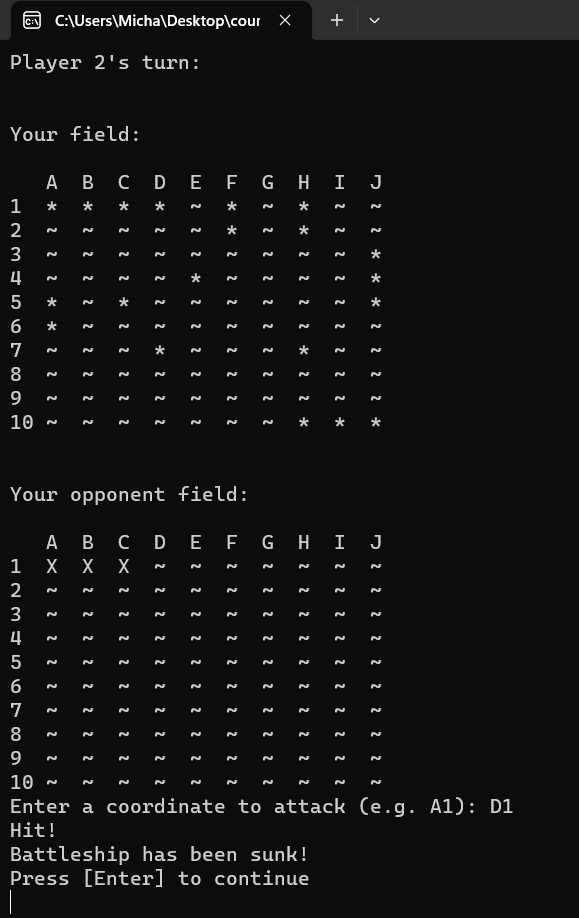


Рисунок 15. Вражеский корабль потоплен

Проверим работу алгоритма определения победителя:



Рисунок 16. Корректная работа алгоритма определения победителя

Заключение

В ходе работы над данным проектом была получена готовая программа, представляющая собой компьютерную реализацию популярной настольной игры «Морской бой». Процесс создания игры позволил ознакомиться с основными принципами программирования при разработке подобных проектов.

В результате работы успешно реализован интерактивный геймплей, который позволяем игрокам размещать свои корабли на игровом поле, а затем проводить атаки на противника, пытаясь разгадать расположение его кораблей. Было создано консольное игровое поле, продуман набор правил, а также представлен многопользовательский режим в рамках одного устройства.

Поставленные цели достигнуты, техническое задание соблюдено. Проект имеет множество возможностей для дальнейшего развития, например добавление графических библиотек или переход на полноценный игровой движок. В будущем возможно добавление искусственного интеллекта для игры «против компьютера», а также онлайн функций, таких как сетевая игра и глобальная таблица рекордов.

Использованная литература

1. Название: «История морского боя»

Автор: Неизвестен

Интернет-адрес: <https://istorygames.info/logicheskie-igryi/istoriya-morskogo-boya.html>

1. Название: «Морской бой»

Автор: Неизвестен

Из журнала «Игры и Игрушки» №5-2017

Интернет-адрес: <https://www.i-igrushki.ru/archive/morskoy-boy5-17.html>

1. Название: «Как научиться играть в морской бой»

Автор: Неизвестен

Интернет-адрес: <https://legkonauchim.ru/igry/kak-nauchitsya-igrat-v-morskoj-boj>

1. Название: «Hasbro’s BATTLESHIP»

Интернет-адрес: <https://store.steampowered.com/app/893050/Hasbros_BATTLESHIP/>

1. Название: «Sea Battle 2»

Интернет-адреса:

* 1. <https://play.google.com/store/apps/details?id=com.byril.seabattle2&hl=ru>
  2. [https://apps.apple.com/ru/app/%D0%BC%D0%BE%D1%80%D1%81%D0%BA%D0%BE%D0%B9-%D0%B1%D0%BE%D0%B9-2/id913173849](https://apps.apple.com/ru/app/морской-бой-2/id913173849)

1. Название: «Battleship»

Интернет-адрес: <https://www.ubisoft.com/ru-ru/game/battleship/battleship>

1. Название: «7 Common Programming Principles That Every Developer Must Follow»

Автор: anuupadhyay

Интернет-адрес: <https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/>

Приложение

Приложение A. Блок-схемы



A.1. Блок-схема общего алгоритма работы функции атаки

Приложение B. Листинги программного кода

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <time.h>
5. #include <stdbool.h>
6. #include <string.h>
7. #define FIELD\_SIZE 10
8. #define NUM\_SHIPS 4
9. typedef char Field[FIELD\_SIZE][FIELD\_SIZE];
10. typedef struct {
11. int x;
12. int y;
13. } Coord;
14. typedef struct {
15. int size;
16. int num\_ships;
17. char\* name;
18. } Ship;
19. void print\_field(Field field, bool show\_ships);
20. void init\_field(Field field);
21. int atack(Field defender\_field, Ship\* ships, int current\_player);
22. bool place\_ship(Field field, Ship\* ship, Coord coord, char direction);
23. bool is\_valid\_position(Field field, Ship\* ship, Coord coord, char direction);
24. Ship\* get\_ship\_by\_size(int size, Ship\* ships);
25. void print\_ships(Ship\* ships);
26. void init\_ships(Ship\* ships);
27. Coord get\_coord\_input(char\* prompt);
28. char get\_direction\_input(char\* prompt);
29. bool is\_coord\_on\_field(Coord coord);
30. Ship\* is\_ship\_sunk(Field field, Coord coord, Ship\* ships, int\* cells\_zero, int N);
31. bool is\_game\_over(Field field);
32. void main\_menu();
33. void game();
34. void about\_game();
35. void about\_dev();
36. void place\_ships(Field field, Ship\* ships, int player);
37. int main() {
38. system("chcp 1251 > null");
39. main\_menu();
40. return 0;
41. }
42. void main\_menu()
43. {
44. int k = 0, t = 1;
45. char c = 0;
46. while (t) {
47. system("cls");
48. printf("\tДобро пожаловать\n\tВыберите раздел (введите число в консоль):\n"
49. "[1] Играть\n[2] Об игре и правила\n[3] О разработчике\n[4] Выйти\n");
50. fflush(stdin);
51. if (scanf("%d", &k) == 1) {
52. while ((c = getchar()) != '\n');
53. switch (k)
54. {
55. case 1:
56. game();
57. break;
58. case 2:
59. system("cls");
60. about\_game();
61. break;
62. case 3:
63. system("cls");
64. about\_dev();
65. break;
66. case 4:
67. return;
68. default:
69. printf("\nНекорректный ввод, пожалуйста проверьте данные и повторите попытку!\n");
70. printf("Нажмите клавишу [Enter] для продолжения:\n");
71. while ((c = getchar()) != '\n');
72. }
73. }
74. else {
75. printf("\nНекорректный ввод, пожалуйста проверьте данные и повторите попытку!\n");
76. printf("Нажмите клавишу [Enter] для продолжения:\n");
77. while ((c = getchar()) != '\n');
78. }
79. }
80. }
81. void about\_game()
82. {
83. char c = 0;
84. printf("Морской бой – это классическая настольная игра, в которой два игрока соревнуются друг с другом,\n"
85. "пытаясь потопить корабли противника на специально размеченном поле.\n"
86. "Игровое поле представляет собой сетку, где вертикальные столбцы обозначают буквы, а горизонтальные строки – цифры.\n\n"
87. "В начале игры каждый игрок размещает корабли на своём поле, где они не видны противнику.\n"
88. "Игроки по очереди должны атаковать клетки на поле противника, при попадании будет выводится 'Hit!', при промахе - 'Miss!'.\n"
89. "Если корабль противника потоплен, будет выведено соответствующее сообщение об уничтожении судна с его названием.\n"
90. "Побеждает тот, кто первым потопит все корабли противника.\n\n"
91. "Нажмите клавишу [Enter] для продолжения:\n");
92. while ((c = getchar()) != '\n');
93. return;
94. }
95. void about\_dev()
96. {
97. char c = 0;
98. printf("Designed by Michael Khudoleev\n"
99. "© Khudoleev M.S., 2023\n\n"
100. "Нажмите клавишу [Enter] для продолжения:\n");
101. while ((c = getchar()) != '\n');
102. return;
103. }
104. void game()
105. {
106. Field p1\_field;
107. Field p2\_field;
108. Ship ships[NUM\_SHIPS];
109. bool p1\_turn = false;
110. char cell = 0, c = 0, direction = 0;
111. char sunk\_ship\_name = 0;
112. int i = 0, j = 0, winner = 0, k = 0;
113. while (1) {
114. system("cls");
115. srand(time(NULL)); // initialize random number generator with current time
116. init\_field(p1\_field);
117. init\_field(p2\_field);
118. init\_ships(ships);
119. place\_ships(p1\_field, ships, 1);
120. init\_ships(ships);
121. place\_ships(p2\_field, ships, 2);
122. init\_ships(ships);
123. // Determine who goes first
124. p1\_turn = (rand() % 2 == 0);
125. while (true) {
126. if (p1\_turn) {
127. printf("Player 1's turn:\n");
128. printf("\n\nYour field:\n\n");
129. print\_field(p1\_field, true);
130. printf("\n\nYour opponent field:\n\n");
131. print\_field(p2\_field, false);
132. winner = atack(p2\_field, ships, 1);
133. }
134. else {
135. printf("Player 2's turn:\n");
136. printf("\n\nYour field:\n\n");
137. print\_field(p2\_field, true);
138. printf("\n\nYour opponent field:\n\n");
139. print\_field(p1\_field, false);
140. winner = atack(p1\_field, ships, 2);
141. }
142. if (winner == 1 || winner == 2) {
143. printf("Congradulations!\n");
144. break;
145. }
146. else {
147. printf("Press [Enter] to continue\n");
148. while ((c = getchar()) != '\n' && c != EOF);
149. system("cls");
150. }
151. // Switch turns
152. if (!winner)
153. p1\_turn = !p1\_turn;
154. }
155. printf("\nВыберите следующее действие (введите число в консоль):\n"
156. "[1] Начать заново\n[2] Вернуться в главное меню\n");
157. fflush(stdin);
158. if (scanf("%d", &k) == 1) {
159. switch (k)
160. {
161. case 1:
162. break;
163. case 2:
164. return;
165. default:
166. printf("\nНекорректный ввод, пожалуйста проверьте данные и повторите попытку!\n");
167. }
168. }
169. else
170. printf("\nНекорректный ввод, пожалуйста проверьте данные и повторите попытку!\n");
171. }
172. }
173. void place\_ships(Field field, Ship\* ships, int player)
174. {
175. Coord coord;
176. int i = 0;
177. char direction = 0, c = 0;
178. for (i = 0; i < NUM\_SHIPS; i++) {
179. Ship\* ship = &ships[i];
180. direction = 0;
181. for (; ship->num\_ships > 0; ship->num\_ships--) {
182. printf("Player %d, place your ships:\n", player);
183. printf("Available ships:\n");
184. print\_ships(ships);
185. printf("Field:\n");
186. print\_field(field, true);
187. printf("Place your %s ship (amount: %d)\n", ship->name, ship->num\_ships);
188. coord = get\_coord\_input("Enter ship's nose coordinate (e.g. A1): ");
189. if (ship->size != 1)
190. direction = get\_direction\_input("Enter ship's direction (up, down, left or right): ");
191. if (!place\_ship(field, ship, coord, direction)) {
192. ship->num\_ships++;
193. printf("Press [Enter] to continue\n");
194. while ((c = getchar()) != '\n' && c != EOF);
195. }
196. system("cls");
197. }
198. }
199. }
200. int atack(Field defender\_field, Ship\* ships, int current\_player) {
201. Coord pCoord;
202. Ship\* destroyed\_ship = NULL;
203. int cells\_zero[4] = { 0 }, i = 0, j = 0;
204. int N = 0;
205. char\* L\_field = \*defender\_field;
206. char cell = 0;
207. char sunk\_ship\_name = 0;
208. pCoord = get\_coord\_input("Enter a coordinate to attack (e.g. A1): ");
209. cell = \*(L\_field + pCoord.y \* FIELD\_SIZE + pCoord.x);
210. while (cell == 'X' || cell == 'O') {
211. printf("You've already attacked that coordinate. Please try again.\n");
212. pCoord = get\_coord\_input("Enter a coordinate to attack (e.g. A1): ");
213. cell = \*(L\_field + pCoord.y \* FIELD\_SIZE + pCoord.x);
214. }
215. if (cell == '~') {
216. printf("Miss!\n");
217. \*(L\_field + pCoord.y \* FIELD\_SIZE + pCoord.x) = 'O';
218. }
219. else {
220. printf("Hit!\n");
221. \*(L\_field + pCoord.y \* FIELD\_SIZE + pCoord.x) = 'X';
222. destroyed\_ship = is\_ship\_sunk(defender\_field, pCoord, ships, cells\_zero, N);
223. if (destroyed\_ship) {
224. printf("%s has been sunk!\n", destroyed\_ship->name);
225. j = cells\_zero[2];
226. for (i = cells\_zero[3]; i <= cells\_zero[1]; i++) {
227. if (j >= 0 && i >= 0 && i <= FIELD\_SIZE - 1) {
228. \*(L\_field + j \* FIELD\_SIZE + i) = 'O';
229. }
230. }
231. j = cells\_zero[0];
232. for (i = cells\_zero[3]; i <= cells\_zero[1]; i++) {
233. if (j <= FIELD\_SIZE - 1 && i >= 0 && i <= FIELD\_SIZE - 1) {
234. \*(L\_field + j \* FIELD\_SIZE + i) = 'O';
235. }
236. }
237. i = cells\_zero[3];
238. for (j = cells\_zero[2]; j <= cells\_zero[0]; j++) {
239. if (i >= 0 && j >= 0 && j <= FIELD\_SIZE - 1) {
240. \*(L\_field + j \* FIELD\_SIZE + i) = 'O';
241. }
242. }
243. i = cells\_zero[1];
244. for (j = cells\_zero[2]; j <= cells\_zero[0]; j++) {
245. if (i <= FIELD\_SIZE - 1 && j >= 0 && j <= FIELD\_SIZE - 1) {
246. \*(L\_field + j \* FIELD\_SIZE + i) = 'O';
247. }
248. }
249. if (is\_game\_over(defender\_field)) {
250. printf("Player %d wins!\n", current\_player);
251. return current\_player;
252. }
253. }
254. return -1;
255. }
256. return 0;
257. }
258. void print\_field(Field field, bool show\_ships) {
259. char\* L\_field = \*field;
261. printf(" ");
262. for (int i = 0; i < FIELD\_SIZE; i++) {
263. if (i <= 8)
264. printf("%c ", 'A' + i);
265. else
266. printf("%c ", 'A' + i);
267. }
268. printf("\n");
269. for (int i = 0; i < FIELD\_SIZE; i++) {
270. if (i <= 8)
271. printf("%d ", i + 1);
272. else
273. printf("%d ", i + 1);
274. for (int j = 0; j < FIELD\_SIZE; j++) {
275. if (show\_ships || \*L\_field == 'X' || \*L\_field == 'O')
276. printf("%c ", \*L\_field);
277. else
278. printf("~ ");
279. L\_field++;
280. }
281. printf("\n");
282. }
283. }
284. void init\_field(Field field) {
285. char\* L\_field = \*field;
286. for (int i = 0; i < FIELD\_SIZE; i++) {
287. for (int j = 0; j < FIELD\_SIZE; j++) {
288. \*L\_field = '~';
289. L\_field++;
290. }
291. }
292. }
293. bool place\_ship(Field field, Ship\* ship, Coord coord, char direction) {
294. char\* L\_field = \*field;
295. if (is\_valid\_position(field, ship, coord, direction)) {
296. if (direction) {
297. for (int i = 0; i < ship->size; i++) {
298. \*(L\_field + coord.y \* FIELD\_SIZE + coord.x) = '\*';
299. if (direction == 'u') {
300. coord.y--;
301. }
302. else if (direction == 'd') {
303. coord.y++;
304. }
305. else if (direction == 'l') {
306. coord.x--;
307. }
308. else {
309. coord.x++;
310. }
311. }
312. }
313. else
314. \*(L\_field + coord.y \* FIELD\_SIZE + coord.x) = '\*';
315. return true;
316. }
317. else {
318. printf("Invalid placement data. Please check it and repeat input!\n");
319. return false;
320. }
321. }
322. bool is\_valid\_position(Field field, Ship\* ship, Coord coord, char direction) {
323. char\* L\_field = \*field;
324. char\* current\_pos = \*field;
325. for (int i = 0; i < ship->size; i++) {
326. current\_pos = L\_field + coord.y \* FIELD\_SIZE + coord.x;
327. if (!is\_coord\_on\_field(coord) || \*(current\_pos) != '~') {
328. printf("[Invalid position error ID: 1; Coords: %d %d]\n", coord.y, coord.x);
329. return false;
330. }
331. if (coord.y == 0) {
332. if (coord.x == 0 && (\*(current\_pos + 1) != '~' || \*(current\_pos + FIELD\_SIZE) != '~' || \*(current\_pos + FIELD\_SIZE + 1) != '~')) {
333. printf("[Invalid position error ID: 2; Coords: %d %d]\n", coord.y, coord.x);
334. return false;
335. }
336. else if (coord.x == FIELD\_SIZE - 1 && (\*(current\_pos - 1) != '~' || \*(current\_pos + FIELD\_SIZE) != '~' || \*(current\_pos + FIELD\_SIZE - 1) != '~')) {
337. printf("[Invalid position error ID: 3; Coords: %d %d]\n", coord.y, coord.x);
338. return false;
339. }
340. else if (coord.x != 0 && coord.x != FIELD\_SIZE - 1 && (\*(current\_pos + 1) != '~' || \*(current\_pos + FIELD\_SIZE) != '~' ||
341. \*(current\_pos + FIELD\_SIZE + 1) != '~' || \*(current\_pos + FIELD\_SIZE - 1) != '~' || \*(current\_pos - 1) != '~')) {
342. printf("[Invalid position error ID: 4; Coords: %d %d]\n", coord.y, coord.x);
343. return false;
344. }
345. }
346. else if (coord.y == FIELD\_SIZE - 1) {
347. if (coord.x == 0 && (\*(current\_pos + 1) != '~' || \*(current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE + 1) != '~')) {
348. printf("[Invalid position error ID: 5; Coords: %d %d]\n", coord.y, coord.x);
349. return false;
350. }
351. else if (coord.x == FIELD\_SIZE - 1 && (\*(current\_pos - 1) != '~' || \*(current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE - 1) != '~')) {
352. printf("[Invalid position error ID: 6; Coords: %d %d]\n", coord.y, coord.x);
353. return false;
354. }
355. else if (coord.x != 0 && coord.x != FIELD\_SIZE - 1 && (\*(current\_pos + 1) != '~' || \*(current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE + 1) != '~' ||
356. \*(current\_pos - FIELD\_SIZE - 1) != '~' || \*(current\_pos - 1) != '~')) {
357. printf("[Invalid position error ID: 7; Coords: %d %d]\n", coord.y, coord.x);
358. return false;
359. }
360. }
361. else if (coord.x == 0 && (\*(current\_pos + 1) != '~' || \*(current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE + 1) != '~' ||
362. \*(current\_pos + FIELD\_SIZE) != '~' || \*(current\_pos + FIELD\_SIZE + 1) != '~')) {
363. printf("[Invalid position error ID: 8; Coords: %d %d]\n", coord.y, coord.x);
364. return false;
365. }
366. else if (coord.x == FIELD\_SIZE - 1 && (\*(current\_pos - 1) != '~' || \* (current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE - 1) != '~' ||
367. \*(current\_pos + FIELD\_SIZE) != '~' || \*(current\_pos + FIELD\_SIZE - 1) != '~')) {
368. printf("[Invalid position error ID: 9; Coords: %d %d]\n", coord.y, coord.x);
369. return false;
370. }
371. else if (coord.x != 0 && coord.x != FIELD\_SIZE - 1 && coord.y != 0 && coord.y != FIELD\_SIZE - 1 &&
372. (\*(current\_pos - 1) != '~' || \*(current\_pos - FIELD\_SIZE) != '~' || \*(current\_pos - FIELD\_SIZE - 1) != '~' ||
373. \*(current\_pos - FIELD\_SIZE + 1) != '~' || \*(current\_pos + 1) != '~' || \*(current\_pos + FIELD\_SIZE) != '~' ||
374. \*(current\_pos + FIELD\_SIZE - 1) != '~' || \*(current\_pos + FIELD\_SIZE + 1) != '~')) {
375. printf("[Invalid position error ID: 10; Coords: %d %d]\n", coord.y, coord.x);
376. return false;
377. }
378. if (direction == 'u') {
379. coord.y--;
380. }
381. else if (direction == 'd') {
382. coord.y++;
383. }
384. else if (direction == 'l') {
385. coord.x--;
386. }
387. else {
388. coord.x++;
389. }
390. }
391. return true;
392. }
393. Ship\* get\_ship\_by\_size(int size, Ship\* ships) {
394. for (int i = 0; i < NUM\_SHIPS; i++) {
395. if (ships[i].size == size) {
396. return &ships[i];
397. }
398. }
399. return NULL;
400. }
401. void print\_ships(Ship\* ships) {
402. for (int i = 0; i < NUM\_SHIPS; i++) {
403. printf("%d %s[size: %d]\n", ships[i].num\_ships, ships[i].name, ships[i].size);
404. }
405. }
406. void init\_ships(Ship\* ships) {
407. ships[0].size = 4;
408. ships[0].num\_ships = 1;
409. ships[0].name = "Battleship";
410. ships[1].size = 3;
411. ships[1].num\_ships = 2;
412. ships[1].name = "Cruiser";
413. ships[2].size = 2;
414. ships[2].num\_ships = 3;
415. ships[2].name = "Destroyer";
416. ships[3].size = 1;
417. ships[3].num\_ships = 4;
418. ships[3].name = "Submarine";
419. }
420. Coord get\_coord\_input(char\* prompt) {
421. Coord coord = { -1, -1 };
422. char c = 0;
423. char test = 0;
424. int i = 0, flag = 0, k = 0;
425. unsigned int len = 0;
426. while (coord.x == -1 || coord.y == -1) {
427. char input[4] = { 0, 0, "P" };
428. flag = 0;
429. i = 0;
430. printf("%s", prompt);
431. fflush(stdin);
432. if (k) {
433. k = 0;
434. while ((c = getchar()) != '\n') {}
435. }
436. while ((c = getchar()) != '\n') {
437. if (i > 3) {
438. flag = 1;
439. break;
440. }
441. if (i == 3) {
442. flag = 3;
443. }
444. input[i] = c;
445. i++;
446. }
447. if (flag == 1 || !input[0]) {
448. printf("Invalid input or coord isn't on the battlefield. Please enter a valid coordinate (e.g. A1). [Input error ID: 1]\n");
449. k++;
450. continue;
451. }
452. if (input[0] >= 'A' && input[0] < 'A' + FIELD\_SIZE && input[1] >= '1' && input[1] < '1' + FIELD\_SIZE) {
453. coord.x = input[0] - 'A';
454. if (input[2] >= '0' && input[2] < '0' + FIELD\_SIZE) {
455. coord.y = (input[1] - '0') \* 10 + (input[2] - '1');
456. break;
457. }
458. else if (flag == 3) {
459. printf("Invalid input or coord isn't on the battlefield. Please enter a valid coordinate (e.g. A1). [Input error ID: 2]\n");
460. continue;
461. }
462. coord.y = input[1] - '1';
463. break;
465. }
466. if (input[0] >= 'a' && input[0] < 'a' + FIELD\_SIZE && input[1] >= '1' && input[1] < '1' + FIELD\_SIZE) {
467. coord.x = input[0] - 'a';
468. if (input[2] >= '0' && input[2] < '0' + FIELD\_SIZE) {
469. coord.y = (input[1] - '0') \* 10 + (input[2] - '1');
470. break;
471. }
472. else if (flag == 3) {
473. printf("Invalid input or coord isn't on the battlefield. Please enter a valid coordinate (e.g. A1). [Input error ID: 3]\n");
474. continue;
475. }
476. coord.y = input[1] - '1';
477. break;
478. }
479. printf("Invalid input or coord isn't on the battlefield. Please enter a valid coordinate (e.g. A1). [Input error ID: 4]\n");
480. continue;
481. }
482. return coord;
483. }
484. char get\_direction\_input(char\* prompt) {
485. while (true) {
486. char input[10];
487. printf("%s", prompt);
488. fflush(stdin);
489. fgets(input, sizeof(input), stdin);
490. fflush(stdin);
491. size\_t len = strlen(input);
492. if (len > 0 && input[len - 1] == '\n') {
493. input[len - 1] = '\0';
494. }
495. if (len >= 2) {
496. switch (input[0]) {
497. case 'u':
498. case 'U':
499. return 'u';
500. case 'd':
501. case 'D':
502. return 'd';
503. case 'l':
504. case 'L':
505. return 'l';
506. case 'r':
507. case 'R':
508. return 'r';
509. }
510. }
511. printf("Invalid direction. Please enter up, down, left or right.\n");
512. }
513. }
514. bool is\_game\_over(Field field) {
515. char\* L\_field = \*field;
516. int i = 0;
517. for (i = 0; i < FIELD\_SIZE \* FIELD\_SIZE; i++) {
518. if (\*L\_field == '\*') {
519. return false;
520. }
521. L\_field++;
522. }
523. return true;
524. }
525. Ship\* is\_ship\_sunk(Field field, Coord coord, Ship\* ships, int\* cells\_zero, int N) {
526. int size = 1, x = 0, y = 0, k = 0, i = 0;
527. int\* tmp = cells\_zero;
528. char\* L\_field = \*field;
529. char\* current\_pos = L\_field + coord.y \* FIELD\_SIZE + coord.x;
530. x = coord.x;
531. while (x <= FIELD\_SIZE - 1) {
532. if (\*(L\_field + coord.y \* FIELD\_SIZE + x) == '\*')
533. return NULL;
534. if (\*(L\_field + coord.y \* FIELD\_SIZE + x) == 'X') {
535. k++;
536. x++;
537. }
538. else
539. break;
540. }
541. \*tmp = coord.y + 1;
542. \*(tmp + 1) = x;
543. x = coord.x;
544. while (x >= 0) {
545. if (\*(L\_field + coord.y \* FIELD\_SIZE + x) == '\*')
546. return NULL;
547. if (\*(L\_field + coord.y \* FIELD\_SIZE + x) == 'X') {
548. k++;
549. x--;
550. }
551. else
552. break;
553. }
554. if (k == 2) {
555. k -= 2;
556. y = coord.y;
557. while (y <= FIELD\_SIZE - 1) {
558. if (\*(L\_field + y \* FIELD\_SIZE + coord.x) == '\*')
559. return NULL;
560. if (\*(L\_field + y \* FIELD\_SIZE + coord.x) == 'X') {
561. k++;
562. y++;
563. }
564. else
565. break;
566. }
567. \*tmp = y;
568. \*(tmp + 1) = coord.x + 1;
569. y = coord.y;
570. while (y >= 0) {
571. if (\*(L\_field + y \* FIELD\_SIZE + coord.x) == '\*')
572. return NULL;
573. if (\*(L\_field + y \* FIELD\_SIZE + coord.x) == 'X') {
574. k++;
575. y--;
576. }
577. else
578. break;
579. }
580. \*(tmp + 2) = y;
581. \*(tmp + 3) = coord.x - 1;
582. k--;
583. return get\_ship\_by\_size(k, ships);
584. }
585. else {
586. \*(tmp + 2) = coord.y - 1;
587. \*(tmp + 3) = x;
588. k--;
589. return get\_ship\_by\_size(k, ships);
590. }
591. return NULL;
592. }
593. bool is\_coord\_on\_field(Coord coord) {
594. return coord.x >= 0 && coord.x < FIELD\_SIZE&& coord.y >= 0 && coord.y < FIELD\_SIZE;
595. }

B.1. Полный листинг исходного кода программы