**«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**

**(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | ДОПУСКАЕТСЯ К ЗАЩИТЕ: | | | | | | | | | | | | | | |
| Факультет | И | |  | Заведующий кафедрой | | | | | |  | | | | И5 | | | | | | |
|  | индекс факультета | |  |  | | | | |  | | | | | | индекс кафедры | | |
| Выпускающая кафедра | И5 | |  | | Скулябина О.В. | | |  | | | |  | | | | | | | |
|  | индекс кафедры | |  | | Фамилия ИО | |  | | | | подпись | | | | | | | | | | |
| Группа | И594 | |  | «\_\_\_\_\_» | |  | | | | | | | | | | 20\_\_\_ г. | | |
|  | индекс группы | |  |  | |  | | | | | | |  | | | |

**отчет**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **о прохождении** | | | учебной | | | | | | | | | | | | | | **практики** | | | | | | | |
| наименование практики | | | | | | | | | | | | | | | | | | | | |
| Кизиловой Валерией Всеволодовной | | | | | | | | | | | | | | | | | | | | |
| Фамилия, имя, отчество обучающегося | | | | | | | | | | | | | | | | | | |
| **обучающегося по**  **направлению/специальности** | | | | | |  | | |  | |  | | | | | | | | | |
| нужное подчеркнуть | | | | | | код | | | |  | | полное наименование направления/специальности | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Руководитель практики:** | | | | | Князьков А.В., профессор каф. И5 | | | | | | | | | | | | | | |
|  | | Фамилия ИО, ученая степень, ученое звание, должность | | | | | | | | | | | | | | | | | | |
| **Срок прохождения практики:** | | | | с | | 28.06.2021 | | | | | | г. |  | по | 25.07.2021 | | | г. | | | |
| **Должность обучающегося на практике:** | | | | | | |  | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Руководитель практики:** | | | |  | | |  |  | | | | |  | | | | | |
|  | | |  | Князьков А.В. | | |  | |  | | |  | |  | | | | |
| Подпись | | |  | Фамилия ИО | | |  | |  | | | |  | |  | | |
| «\_\_\_» |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |  | 20\_\_г. |  |  | |  |  | | | | |  |  | | |

САНКТ-ПЕТЕРБУРГ

2021 г.

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc78110955)

[ВВЕДЕНИЕ 3](#_Toc78110956)

[1 Постановка задачи 4](#_Toc78110957)

[2 Общие компоненты программы 5](#_Toc78110958)

[3 Клиент 7](#_Toc78110959)

[3.1 Описание клиента 7](#_Toc78110960)

[3.2 Класс «ClientController» 8](#_Toc78110961)

[3.3 Класс «GameClientController» 8](#_Toc78110962)

[3.4 Классы «gameObj» и «ship» 9](#_Toc78110963)

[4 Сервер 11](#_Toc78110964)

[4.1 Описание сервера 11](#_Toc78110965)

[4.1 Класс «ServerController» 11](#_Toc78110966)

[4.2 Класс «GameLogicController» 12](#_Toc78110967)

[5 Графическое описание программы 13](#_Toc78110968)

[5.1 Класс «graphicEnv» 13](#_Toc78110969)

[5.2 Функции «scale\_x» и «scale\_y» 13](#_Toc78110970)

[5.3 Классы «Entity» и «Button» 13](#_Toc78110971)

[6 Внешний вид запущенной программы 15](#_Toc78110972)

[6.1 Меню 15](#_Toc78110973)

[6.2 Помощь 16](#_Toc78110974)

[6.3 Игровой процесс 16](#_Toc78110975)

[5.8 Выход 19](#_Toc78110976)

[ЗАКЛЮЧЕНИЕ 21](#_Toc78110977)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22](#_Toc78110978)

[ПРИЛОЖЕНИЕ А. Описание электронного носителя 23](#_Toc78110979)

# ВВЕДЕНИЕ

Игры — это большая часть индустрии информационных технологий. Программирование игр полезно как тем, кто любит проводить в них время, так и тем, кто их непосредственно создает, поскольку для программиста игры — это поле для решения огромного числа задач. Например, написание серверной игры для программиста означает, что нужно разобраться с принципами работы и создания серверов, но также такая работа не может обойтись без понимания работы с графикой, движением, памятью, обработкой событий и множеством других распространенных задач.

Таким образом, цель данной работы — создать серверную игру для двух пользователей «Морской бой» с использованием библиотек SDL2 .

# 1 Постановка задачи

Задача проекта по учебной практике — разработать онлайн-игру «Морской бой».

Игра «Морской бой» представляет собой аналогичную настольную игру, но в онлайн-формате. В данном проекте будет использован вариант игры с полем 10 на 10 клеток и четырьмя типами кораблей, однако можно использовать и другое поле, и, соответственно, другое количество типов кораблей. Игра делится на несколько этапов: расстановка кораблей, непосредственно бой и подведение итогов. Главное условие игры — игроки не знают расположение кораблей друг друга.

Рассмотрим расстановку кораблей. Всего есть четыре типа кораблей по количеству занимаемых клеток — от одной до четырех. Каждому игроку дается четыре корабля по одной клетке, три корабля по две клетки, два корабля по три клетки и один в четыре клетки. Каждый корабль можно повернуть. Между кораблями должна быть минимум одна свободная клетка, корабли не могут пересекаться между собой.

Во время боя игроки выбирают на поле противника точки, в которые будет производиться обстрел. В случае успешного попадания в корабль противника, игрок стреляет снова, пока не промажет. Если игрок не попадает по кораблю, то ход передается другому игроку. После того, как у одного из игроков закончатся корабли, игра заканчивается.

Для обеспечения честности игры информация о расположении кораблей игроков хранится на сервере. Ввиду этого будет использована библиотека SDL\_Net 2.0, так как при помощи неё можно реализовать сервер.

Чтобы игра обладала интерфейсом, понятным пользователю, необходимо выводить текст на экран — используем для этого библиотеку SDL\_ttf 2.0.

# 2 Общие компоненты программы

Поскольку стоит задача создать серверную игру, будут описаны клиент и сервер. При их создании будут задействованы некоторые основные компоненты, к примеру, классы и перечисления.

Начнем с перечислений. Постановка задачи накладывает такое свойство, как наличие множества состояний ячейки, а наличие сервера предполагает наличие пакетов сообщений. Поэтому большую роль в разработке играют именно перечисления, поскольку с их помощью можно гораздо более удобно обрабатывать поле. Так, например, перечисление states содержит возможные состояния ячейки:

* emptyCell — ячейка пуста,
* shipSize1, shipSize2, shipSize3, shipSize4 — в ячейке расположен корабль с определенным количеством ячеек,
* shipDenied — в клетку нельзя поставить корабль,
* shipDamaged — в клетке находится поврежденная часть корабля,
* shipDestroyed — в клетке находится часть «убитого» корабля,
* shotMissed — по клетке был произведен выстрел, но она пуста.

Перечисления сообщений описывают возможные сообщения, которые может отправить клиент или сервер.

* emptyPackage и endOfPackages — сообщения, свойственные и клиенту, и серверу. Обозначают пустой пакет или конец пакетов,
* requestPlaceShip — сообщение с запросом клиента о расположении корабля,
* requestStrikeInfo — сообщение с запросом клиента об информации о выстреле,
* requestMessages — запрос клиента о сообщении,
* requestIndex — запрос клиента об индексе игрока,
* requestTurn — запрос клиента о повороте корабля,
* turnInfo, fieldInfo — сообщения от сервера с информацией о повороте корабля и с информацией о поле,
* strikeSuccess, strikeMissed — сообщения от сервера об успехе выстрела,
* setIndex — сообщение сервера с индексом игрока,
* infoBothConnected — информация от сервера о том, оба ли игрока установили соединение,
* battleBegins, battleEnds — информация о начале или конце сражения,
* timeToSendPackages — специальное значение для пересылки пакетов.

Оформить эти сообщения в пакет можно с помощью структуры. Структура Package имеет два поля: int data[1024] с самими данными и Message message с сообщением пакета.

При создании и клиента, и сервера понадобится класс обработки пакетов PackageController. Он обладает двумя полями типа queue<Package> sendPackagesQueue[2] и receivedPackagesQueue[2], описывающими очереди отправленных и принятых пакетов для каждого клиента. Методы данного класса реализуют основные действия при обработке пакетов: void addPackageToSendQueue(Package package, int index) добавляет клиенту в очередь отправленных пакетов еще один пакет, void addPackageToReceivedQueue(Package package, int index) делает то же самое, но добавляет в очередь полученных пакетов. Метод void sendPackages(TCPsocket receiver, int index) отправляет пакеты получателю, а void receivePackages(TCPsocket sender, int index) получает пакеты от отправителя. Package getFrontReceivedPackage(int index) добавляет вперед очереди полученных пакетов полученный пакет.

# 3 Клиент

# 3.1 Описание клиента

Клиент — программа, предназначенная для непосредственного взаимодействия с пользователем. Он запрашивает и ожидает ответы от сервера, ответы в свою очередь представляют собой пакет сообщений. Разберем подробнее, что происходит в главной функции клиента и в функциях, относящимся к нему.

В главной функции инициализируются необходимые библиотеки, шрифт, а также вызывается функция меню для взаимодействия с меню int menu(SDL\_Renderer\* ren, TTF\_Font\* font). Функция реализует три пункта меню: «играть», «помощь» и «выйти». Далее при помощи обработчика событий SDL\_PollEvent() происходит обработка выбора пунктов меню. Можно выбрать пункт мышкой, можно нажав соответствующую цифру пункта меню на клавиатуре. Возвращает функция 0 в случае, если надо остаться в меню, и 1, если из меню нужно выйти (например, при окончании игры или выборе пункта «выйти»).

При выборе пункта «играть» вызывается функция int start(SDL\_Renderer\* renderer, TTF\_Font\* font), отвечающая за старт игры. Для начала устанавливается соединение с сервером при помощи объектов классов ClientController и PackageController. IP-адреса указаны для localhost[1], однако есть возможность изменить адрес на иные. В случае успешного соединения создается объект игрового контроллера GameClientController и загружаются необходимые текстуры для игры (фон, поля, ячейки кораблей в трех состояниях). Запускается игровой цикл while(!done), в котором вложены обработчик событий while(SDL\_PollEvent(…)) [2, c. 34, 3] и обработчик пакетов (их формирование и отправка производится средствами класса PackageController), а также процесс прорисовки элементов поля.

В программе клиента содержатся функции для прорисовки поля void drawField(SDL\_Renderer\* renderer, SDL\_Rect surfSize, SDL\_Texture\* txtr1, SDL\_Texture \*txtr2) и вспомогательные глобальные переменные для обозначения габаритов экрана, количества кнопок, необходимых цветов и так далее.

Для полноценного описания клиента потребуются несколько основных и вспомогательных классов, среди которых классы ClientController, GameClientController.

## 3.2 Класс «ClientController»

Класс ClientController — контроллер для управления клиентом, обладает полями и методами, дающими информацию о подключении игроков. Содержит приватные поля TCPsocket serverSocket с информацией о сокетах [4], IPadress ip с IP-адресом игрока и флаги: bool connected — флаг подключения, bool bothPlayersConnected (публичное поле) — флаг подключения обоих игроков.

Рассмотрим методы. Методы данного класса реализуют основные действия, необходимые при работе с клиентом. В конструкторе без параметров задаются начальные значения полей. Метод void connect(const char\* serverIP, Uint16 port) обеспечивает подключение клиента к серверу, принимая на вход значение IP-адреса сервера и значение порта. Метод TCPsocket getServerSocket() предназначен для установки значения поля serverSocket. void closeConnection() отключает соединение с сервером. Все описанные методы публичные.

## 3.3 Класс «GameClientController»

Данный класс отвечает за обработку игровых действий клиента обоих игроков. В связи с разделением игры на три этапа (расстановка кораблей, бой и подведение итогов) вводится перечисление currentGameStates с соответствующими этапам значениями shipPlacement, battle и results. Также вводится переопределение 4 как maxAmount — максимальное количество кораблей, а также переменные, определяющие длину и ширину экрана SCREEN\_WIDTH и SCREEN\_HEIGHT.

Перейдем к непосредственно описанию класса. Все его поля и методы публичные. Поле int playerFields[2][10][10] описывает поля игроков для передачи на сервер, int playerIndexOnServer содержит в себе индекс игрока, int gameState содержит информацию о состоянии игры (опять же, соответствующе трем этапам игры). ship\* grabbedShip — это указатель на переменную типа ship, т. е. корабль, в данный момент выбранный игроком, std::vector <ship> ships[maxAmount] — вектор, хранящий в себе все корабли. SDL\_Rect dest[maxAmount] — массив положений кораблей.

Конструктор класса принимает на вход SDL\_Renderer \*ren. В самом конструкторе происходит инициализация полей класса, в частности установка состояния игры как shipPlacement, загрузка текстур кораблей в массив и установка стартовых положений для них.

Метод Package createShotPackage(int x, int y) создает пакет с информацией о том, куда стреляет игрок.

void proceedResponse(Package response) обрабатывает полученный на вход пакет, в зависимости от сообщения в пакете выполняя определенные действия — подавая информацию о поле, отмечая ячейку поле как с поврежденной частью корабля или с промахом, изменяя статус игры в зависимости от этапа.

Package handleMouseEvent(SDL\_Event \_event) используется для обработки событий мыши, в частности для перетаскивания корабля и для выстрелов. Производится проверка состояния игры и в зависимости от этого действия мышки воспринимаются как расстановка или как попытка произвести выстрел

## 3.4 Классы «gameObj» и «ship»

Эти классы имеет смысл обозревать вместе. gameObj — класс игрового объекта, определяющий его основные свойства. Класс «ship» наследуется от класса gameObj и задает основные свойства для управления кораблем.

Так, в gameObj защищенные поля SDL\_Renderer\* render в качестве основного рендерера, SDL\_Surface\* surf в качестве основной поверхности, SDL\_Texture\* txtr в качестве основной текстуры. SDL\_Rect place — поле с координатами корабля, int cellX и cellY описывают координаты текущей клетки корабля. Имеется чисто виртуальный метод отображения virtual void image().

Класс ship расширяет эти свойства с помощью защищенных полей SDL\_Event event для обработки событий, int cellsValue с количеством ячеек корабля, bool grabbingFlag — флаг, указывающий схвачен ли корабль, bool rotationFlag — показывающий, повернут ли корабль. SDL\_Rect initialPlace хранит координаты, показывающие изначальное положение корабля при его передвижении.

Методы описывают основные механики, которыми обладает корабль. В конструкторе без параметров инициализируются поля, в деструкторе очищаются поверхность и текстура. Метод loadSkin(SDL\_Renderer\* ren) осуществляет загрузку текстуры корабля в зависимости от количества его ячеек. void saveInitialPos() сохраняет текущую позицию как новую изначальную, void restPosition() возвращает корабль на заданную изначальную позицию initialPlace и приводит корабль в исходную ротацию. Метод setPlace(SDL\_Rect dest) устанавливает переданные координаты как текущее место, а setPlace() устанавливает координаты в зависимости от ячейки в которой находится корабль. SDL\_Rect getPlace() возвращает информацию о текущих координатах корабля. void handleEvent(SDL\_Event evt) обрабатывает события корабля и изменяет его состояния в зависимости от них. Методы int\* getCell() и int getCellsValue() позволяют получить информацию, в первом случае о координатах ячейки, во втором о количестве ячеек у корабля. Соответствующие им методы void setCell(int \_cellX, int \_cellY) и void setCellsValue(int cellsValue) устанавливают эти значения. Реализованы два метода для вращения корабля — void rotate() и bool getRotation(), устанавливающие и возвращающие значение флага rotationFlag. Метод SDL\_Texture\* getShipTexture() передает текстуру корабля.

# 4 Сервер

# 4.1 Описание сервера

При реализации сервера используется единственная главная функция. В ней происходит инициализация библиотек, создание контроллеров непосредственно самого сервера и игрового процесса, вычисление которого происходит на сервере. Обработка происходит при помощи таймера и потоков пакетов, которые обрабатываются в общем цикле, пока соединение с сервером не понадобится разорвать (в случае победы или выхода из игры). В целом обработку сервером можно разделить на три этапа по этапам игры: расстановка кораблей, игровой процесс и окончание игры.

## 4.1 Класс «ServerController»

Класс «ServerController» обладает полями и методами, необходимыми для управления сервером и доступа к его свойствам. Все поля и методы являются публичными. TCPsocket server описывает информацию о сетевом сокете [2], vector<TCPsocket> players — вектор, содержащий в себе сетевые сокеты игроков. IPadress ip хранит IP-адрес соединения, SDLNet\_SocketSet socketSet — указатель на несколько сокетов для удобства работы с ними [5]. Далее есть два флага — bool done и bool bothPlayersConnected, свидетельствующий о подключении обоих игроков к серверу.

Методы класса обеспечивают основные действия с сервером. Конструктор класса без параметров запускает сервер, инициализирует поля. Метод void getConnections() устанавливает соединение игроков и повторяет запросы, пока оба игрока не присоединятся к игре. int printPlayerInfo(int playerIndex) выводит в консоль информацию о клиенте: его номер (то есть номер текущего игрока), IP-адрес, порт. void closeConnection(int playerIndex) останавливает соедниение для конкретного клиента, заданного переменной playerIndex, vector<TCPsocket> getPlayersSockets() возвращает информацию о сокетах игроков в виде вектора типа TCPsocket.

## 4.2 Класс «GameLogicController»

Этот класс отвечает за проработку игровой логики на сервере, то есть все игровые расчеты проводятся тут.

Полями класса являются int playerFields[2][10][10], описывающая поле в текущий момент игры, и int shipsPlaced[2] для обозначения количества расположенных игроками кораблей.

Конструктор без параметров проводит инициализацию поля начальными состояниями (ноль — пустая клетка), устанавливает так же изначальное количество кораблей на поле для каждого игрока — соответственно, тоже ноль. Метод Message proceedShot(int playerIndex, int x, int y) производит попытку выстрела по конкретной ячейке и в зависимости от состояния этой ячейки возвращает сообщение об успешном выстреле или о промахе. Также может быть возвращен пустой пакет сообщений. Package proceedRequest(Package package) обрабатывает пакеты сообщений. На вход поступает пакет сообщений, который обрабатывается при помощи оператора switch. Возвращается, в свою очередь, либо пустой пакет, либо ответ. В методе описаны все соответствующие перечислению сообщений Message случаи, в том числе расстановка кораблей с учетом правил (между кораблями должна быть минимум одна клетка), запрос информации о поле, запрос о возможности выстрела, поворот, запрос сообщений.

# 5 Графическое описание программы

## 5.1 Класс «graphicEnv»

Класс графического интерфейса предназначен для работы с графической частью игры, например, менять размер окна. Класс можно дополнять иными свойствами.

Обладает приватными полями SDL\_Texture\* txtr, SDL\_Surface\* surf, SDL\_Rect dest, bool win\_flag. Это текстура, поверхность, координаты чего-либо. bool win\_flag — это флаг размера окна, показывающий развернуто окно на весь экран или нет.

Конструктор обеспечивает инициализацию переменных, деструктор удаляет текстуру и поверхность. Имеется метод получения координат SDL\_Rect getDest(void), методы работы с окном void invertWindowFlag(void) и bool getWindowFlag(void) меняющие свойство режима окна и возвращающие информацию о нем.

## 5.2 Функции «scale\_x» и «scale\_y»

Они описаны в части программы, отвечающей за отрисовку игры и позволяют масштабировать изображение при изменении размера экрана. За «эталонный» габарит экрана принято разрешение 720 на 540 пикселей.

## 5.3 Классы «Entity» и «Button»

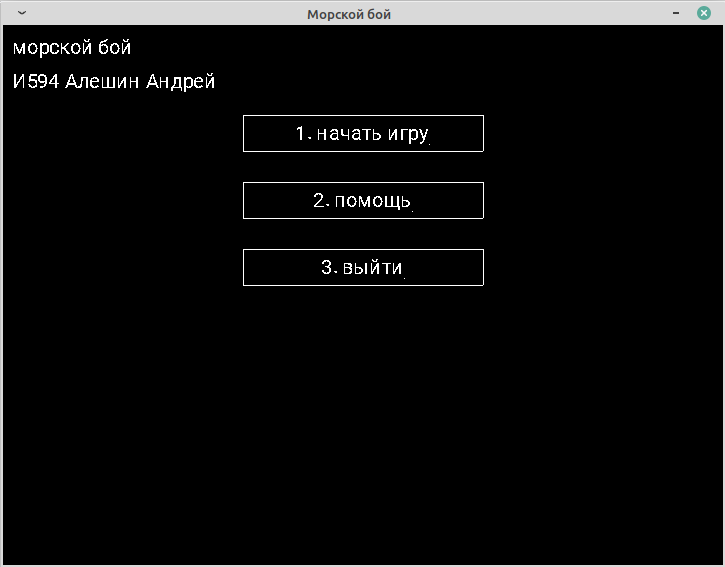
Классы, обеспечивающие корректное отображение окружения программы клиента. Класс Entity является основным. Его поля int x, y, width, height описывают координаты и размер. Методы void setPosition(int X, intY), void setSize(int w, int h) устанавливают значения положения и размера, инициализируя соответствующие поля, а методы int getX(), int getY(), int getWidth(), int getHeight() дают доступ к этим полям. SDL\_Rect getRect конвертирует поля класса в переменную типа SDL\_Rect и возвращают ее.

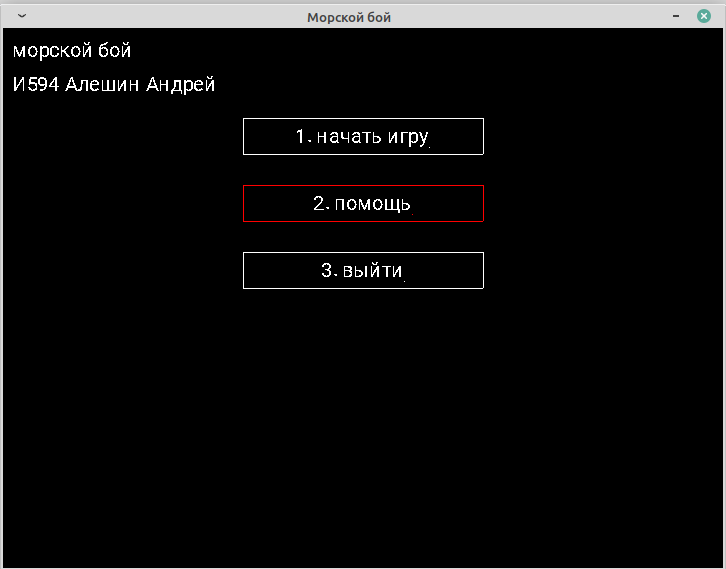
Класс Button наследуется от класса Entity, описывает кнопку. У него есть дополнительное поле string text, в котором хранится текст текущей кнопки. Конструктор класса без параметров инициализирует строки и положения базовыми величинами (нули и пустая строка), конструктор с параметрами принимает на вход значения int X, int Y, int w, unt h, string Text и инициализирует ими соответствующие поля. Методы void setText(string text) и string getText() устанавливают текст и возвращают содержимое соответствующего поля класса. Метод void draw(SDL\_Renderer renderer, SDL\_Color color, TTF\_Font font, SDL\_Color textColor) прорисовывает кнопку. Для его реализации необходимы функции void drawRect(SDL\_Renderer renderer, int x1, int y1, int w, int h, SDL\_Color color) и void drawText(SDL\_Renderer\* renderer, TTF\_Font\* font, int x, int y, const char\* text, SDL\_Color color), рисующие прямоугольник и выводящие текст на экран.

# 6 Внешний вид запущенной программы

## 6.1 Меню

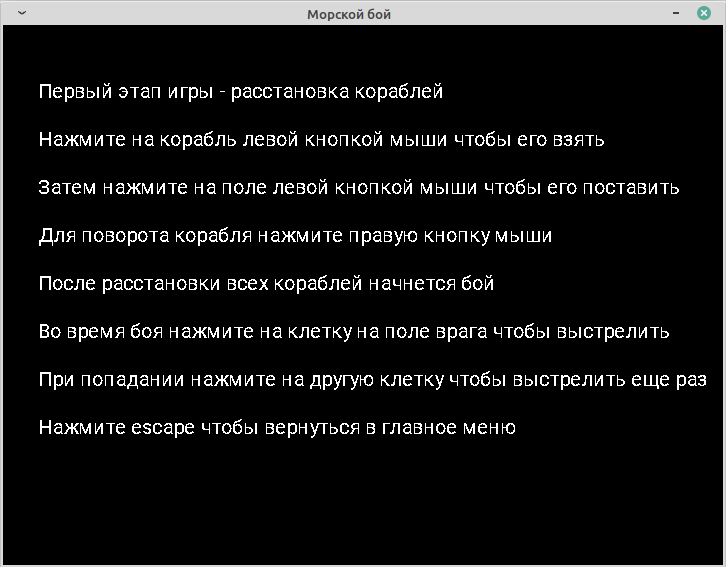
При запуске программы пользователю показывается меню игры с пунктами «начать игру», «помощь», «выйти». Изображения работы меню представлено на рисунках 1 и 2.

Рисунок 1 — внешний вид заставки

Рисунок 2 — процесс выбора пункта меню

## 6.2 Помощь

При выборе пункта меню «помощь» выводится справка об игре с содержанием ее правил. Это представлено на рисунке 3.

Рисунок 3 — вывод правил игры на экран

## 6.3 Игровой процесс

При выборе пункта меню «начать игру» пользователям выводится поле. У одного игрока поле справа, у другого игрока поле слева, как это показано на рисунке 4. Если соединение все еще устанавливается, то пользователь видит сообщение о соединении с сервером, что продемонстрировано на рисунке 5.

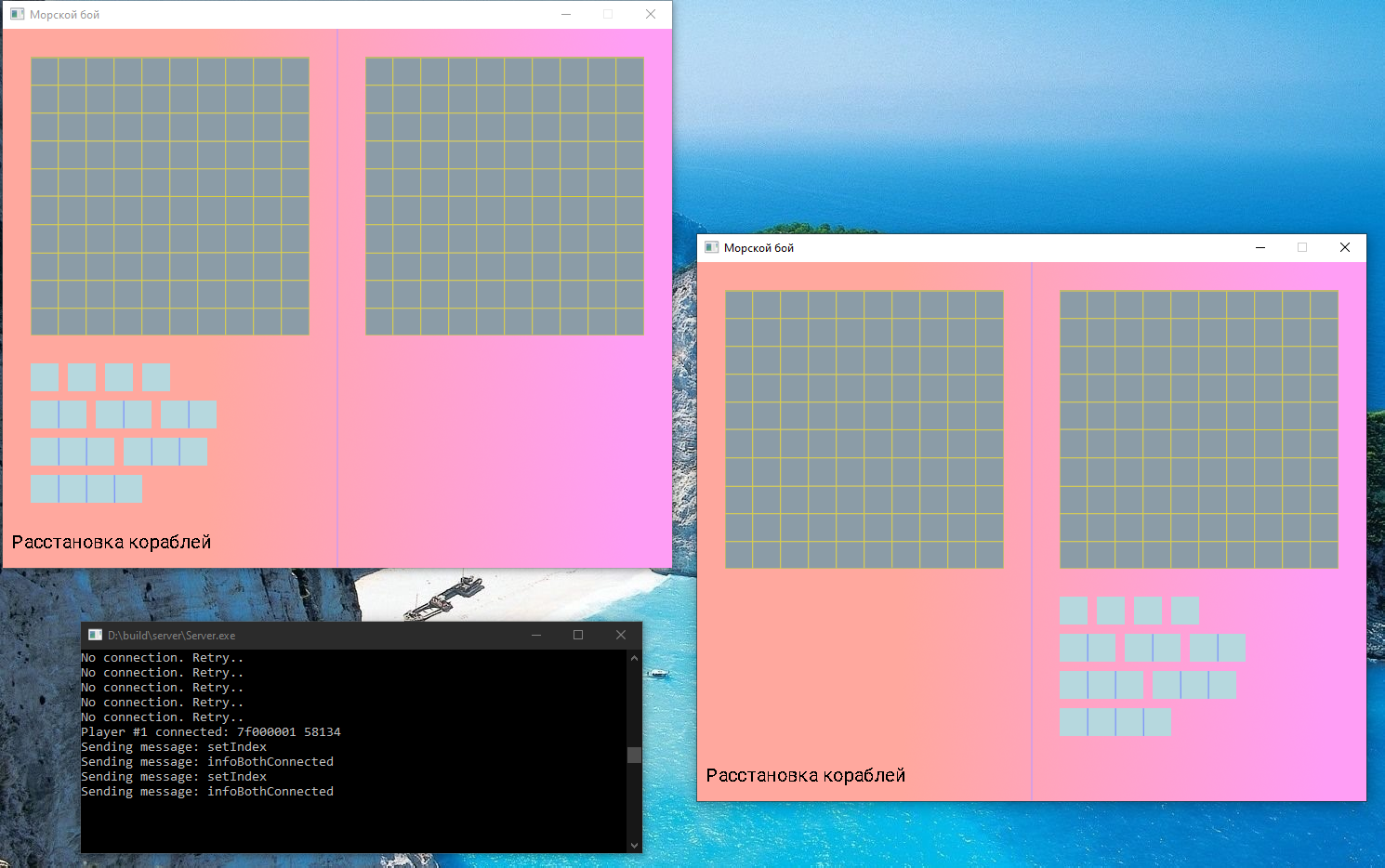


Рисунок 4 — игровые среды игроков, поток сообщений сервера

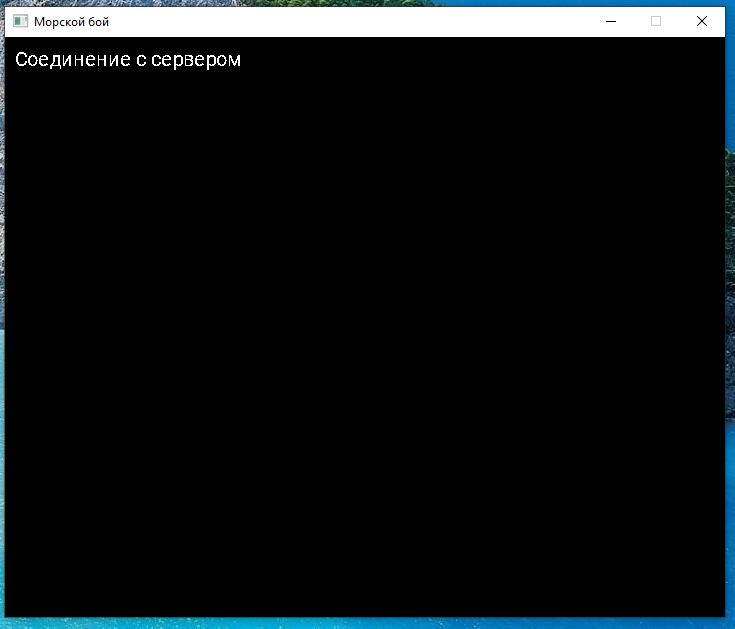


Рисунок 5 — демонстрация продолжительного соединения с сервером

Игра начинается с расстановки кораблей. Процесс расстановки кораблей показан на рисунках 6, 7. На них продемонстрировано перетаскивание кораблей из области «хранения» на поле, поворот, а также поток сообщений сервера

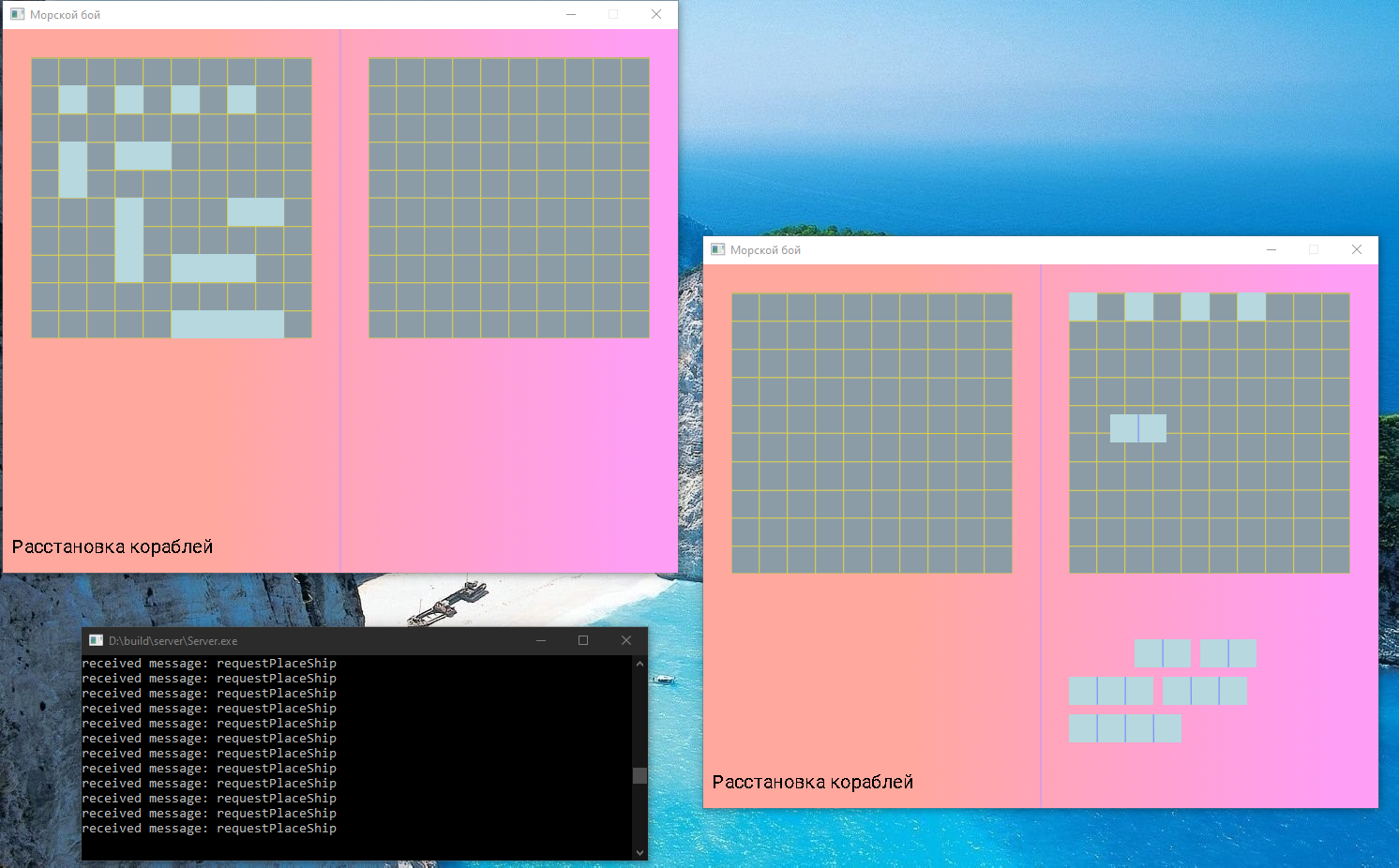


Рисунок 6 — процесс перетаскивания корабля на поле

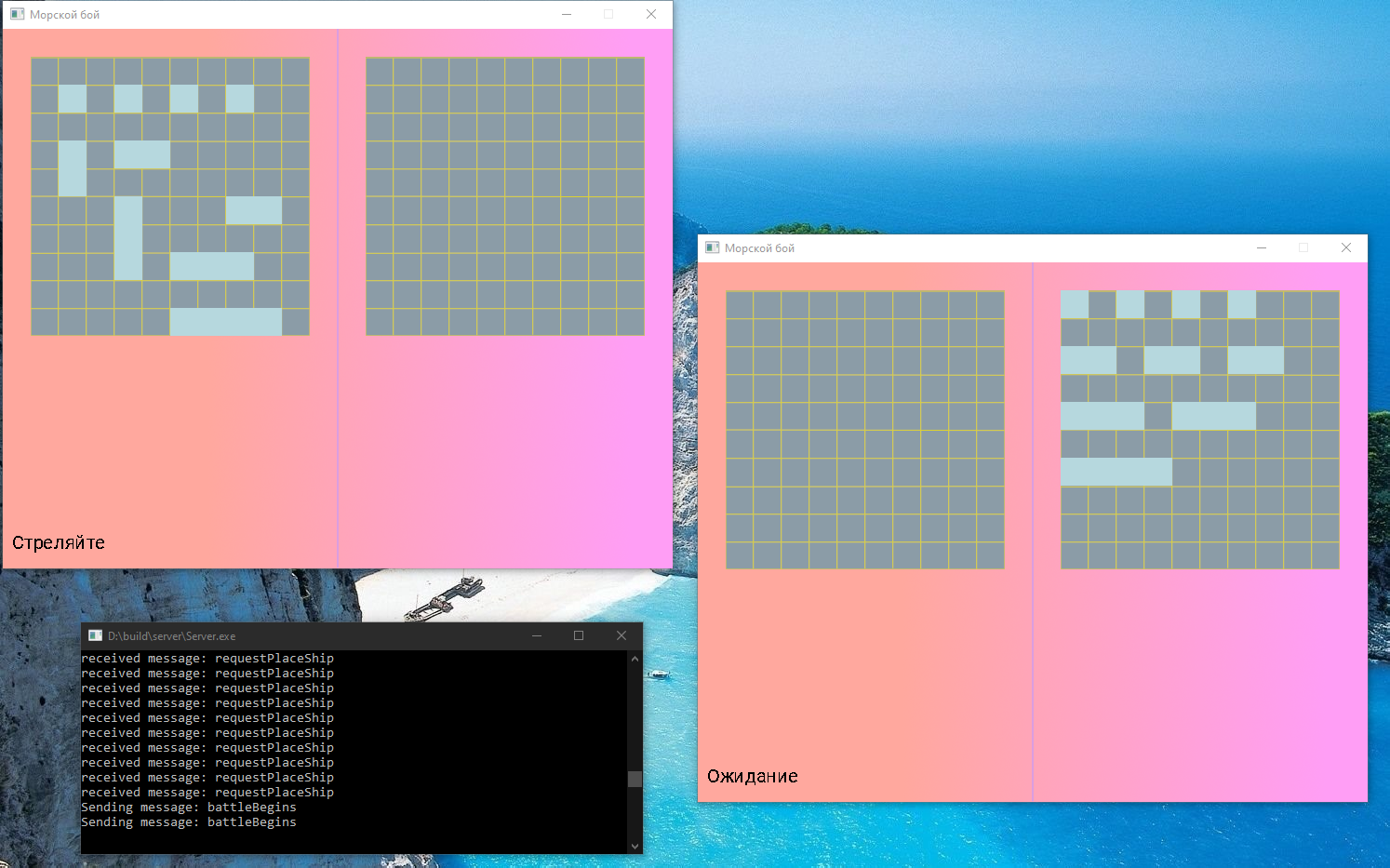


Рисунок 7 — окончание этапа расстановки кораблей

Процесс выстрелов показан на рисунке 8. Сначала стреляет один игрок, после его промаха (точка на карте) ход передается другому игроку.

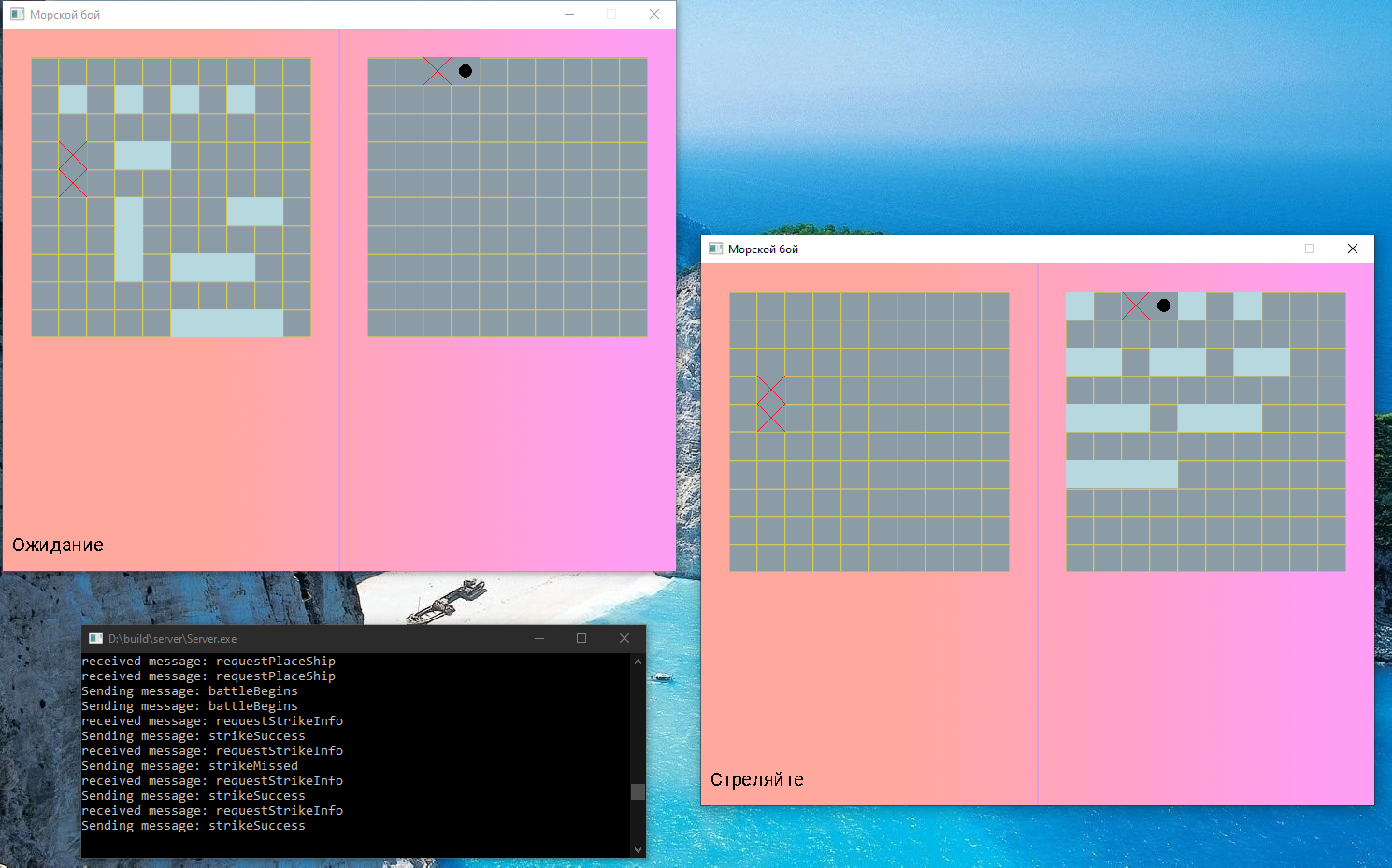


Рисунок 8 — выстрелы: удачные (крестики) и промахи (точки)

Игра заканчивается, когда у одного из игроков заканчиваются корабли. При окончании игры показываются сообщения, продемонстрированные на рисунке 9.

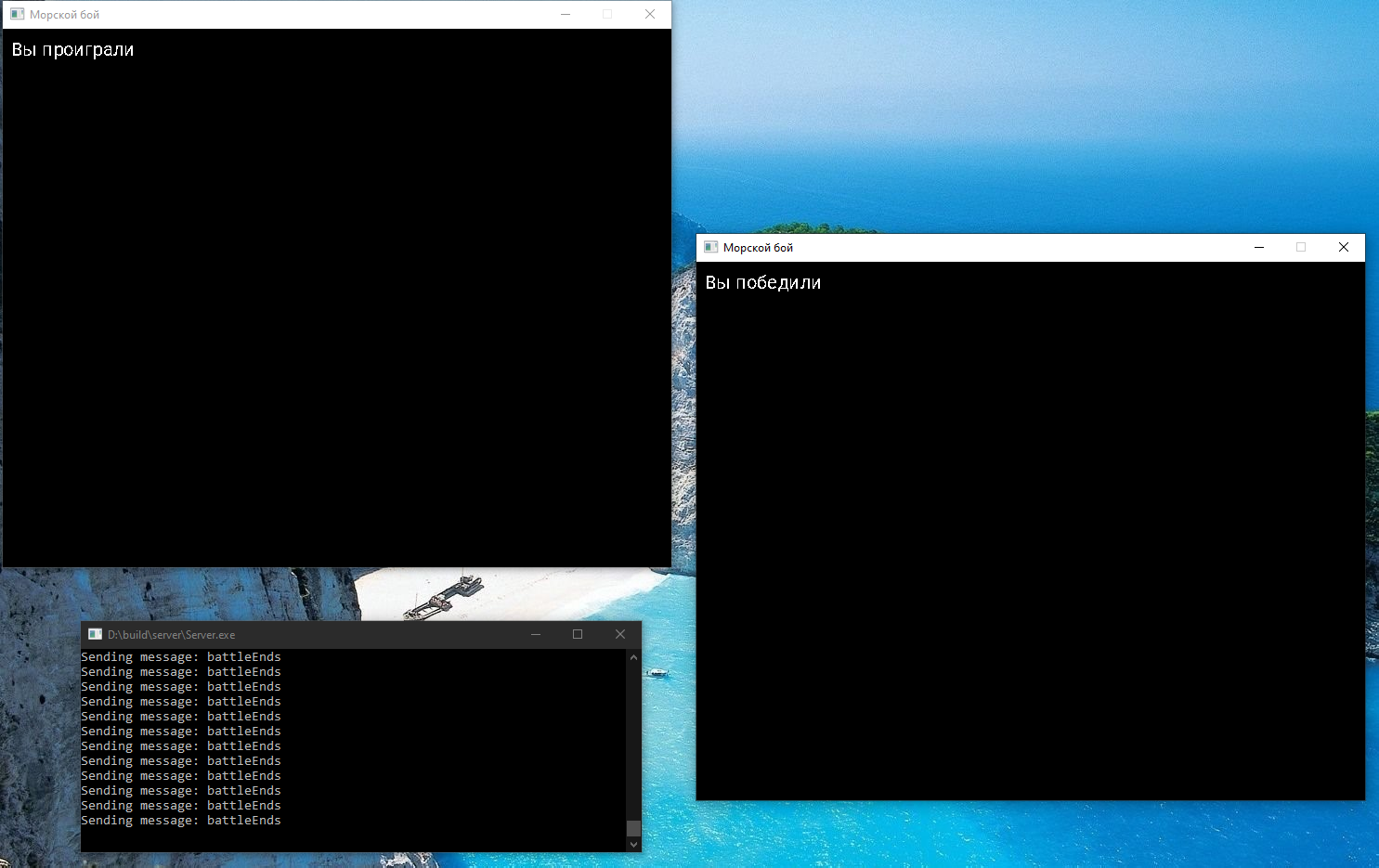


Рисунок 9 — конец игры

## 6.5 Выход

При выборе пункта меню «выйти» работа программы завершается (см. рисунок 16).

  
Рисунок 10 — завершение работы программы

Полные исходные тексты приведены в электронном носителе, приведенном в приложении А.

# ЗАКЛЮЧЕНИЕ

В ходе разработки собственной игры было получено существенное понимание принципов работы с графикой, ООП, серверами, усвоены принципы формирования пакетов сообщений и их обработка. Объектно-ориентированное программирование действительно является очень важным элементом разработки подобных программ, в которых есть множество однотипных сущностей с каким-либо общим поведением, составляющими и целями. Парадигмы объектно-ориентированного программирования также позволяют более рационально использовать код, что, конечно, экономит и время, и силы, и вносит более очевидную логику в процесс.

Также был сделан вывод, что разработка приложений с подобным интерфейсом имеет смысл осуществляться именно снизу-вверх, поскольку крупную задачу такого рода проще собрать из нескольких поэтапно выполненных фрагментов.

Таким образом, была успешно реализована серверная игра «Морской бой» для двух пользователей. Поставленные задачи выполнены. В дальнейшем возможна реализация дополнительных механик и расширение ныне существующих возможностей игры.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Hostinger Руководства [Электронный ресурс]. — URL: https://www.hostinger.ru/ (дата обращения: )
2. Гущин, А. Н. Применение библиотеки SDL для разработки программ на языке C [Текст]: учебное пособие для вузов / А. Н. Гущин. - СПБ.: [б. и.], 2014. - 118 c.
3. SDL Wiki [Электронный ресурс]. — URL: <https://wiki.libsdl.org/> (дата обращения: 22-01-2021).
4. SDL\_net 1.2.7 [Электронный ресурс]. — URL: https://www.libsdl.org/projects/SDL\_net/docs/SDL\_net.html (дата обращения: 20-01-2021).
5. Server-Gu.ru Мониторинг, настройка и обслуживание Linux-серверов [Электронный ресурс]. — URL: https://server-gu.ru/ (дата обращения: 23-01-2021).

# ПРИЛОЖЕНИЕ А. Описание электронного носителя

Папка проекта имеет название Kizilova\_VV\_I594\_Practice\_MorskoyBoy. Она содержит в себе 5 подпапок

Папка client предназначена для реализации контроллеров, связанных с клиентом. Так, в подпапках ClientController и GameClientController находятся одноименные файлы с расширениями .h и .cpp.

В папке server находятся файлы, непосредственно описывающие реализацию сервера, в частности подпапки GameLogicController и ServerController, в которых находятся заголовочные файлы и реализации с одноименными названиями и расширениями .h и .cpp.

Директория common содержит в себе файл Package.h и подпапку PackageController с файлами реализации класса PackageController с расширениями .h и .cpp.

Папка gameObjects хранит два заголовочных файла: gameObj.h и graphicEnv.h, а также подпапку button с файлами Entity.h, Entity.cpp, Button.h, Button.cpp и подпапку res с необходимыми для игры текстурами и шрифтом.

main example содержит два файла: clientMain.cpp для реализации main-файла клиента и server.cpp с реализацией main-файла сервера.