Государственное учреждение образования

«Институт повышения квалификации и переподготовки в области технологий информатизации и управления» БГУ

Экономической кибернетики и информатики

ДОПУЩЕН К ЗАЩИТЕ:

Зав.кафедрой

Н.Л. Черкас

« » 20 19

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

по дисциплине «Средства визуального программирования приложений» на тему: «Разработка компьютерной сетевой игры «Морской бой» с помощью технологии Windows Forms платформы .Net Framework»

Выполнил слушатель группы ПОИС1709в1

Пузанов Кирилл Владимирович

(подпись, дата)

Научный руководитель

старший преподаватель

Буславский Александр Андреевич

(подпись, дата)

Проект защищен с оценкой

Минск 2019

**Бланк для оформления листа задания курсового проекта**

Государственное учреждение образования

«Институт повышения квалификации и переподготовки в области технологий информатизации и управления» БГУ

Экономической кибернетики и информатики

|  |  |
| --- | --- |
| Специальность Программное  обеспечение информационных  систем  Группа ПОИС1709в1 | УТВЕРЖДАЮ  Заведующий кафедрой  Н.Л.Черкас  « » 20 г. |

Задание

на курсовое проектирование

Слушателю Пузанову Кириллу Владимировичу 1 Тема проекта: Разработка компьютерной сетевой игры «Морской бой» с помощью технологии Windows Forms платформы .Net Framework 2 Срок сдачи законченного проекта « 3 » июня 2019г.

1. Исходные данные по проекту: (в краткой форме приводится информация, являющаяся вводной для создаваемого проекта)

[Интегрированная среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения разработки Microsoft Visual Studio, технология интеллектуального клиента Windows Forms для платформы .Net Framework

1. Содержание расчетно-пояснительной записки:

Введение, 1. Типы архитектур информационных систем, 1.1 Понятие архитектуры системы, 1.1.1 Архитектура файл-сервер, 1.1.2 Архитектура клиент-сервер, 1.1.3 Многоуровневая архитектура, 1.2 Технология интеллектуального клиента Windows Forms, 1.3 Многопоточность, 1.4 Делегаты, 2 Проектирование клиент-серверного приложения с помощью технологии Windows Forms, 2.1 Структура игрового приложения «Морской бой», 2.1.1 Формирование расстановки кораблей игрока, 2.1.2 Реализация взаимодействия между клиентом и сервером, 2.1.3 Запуск и тестирование работы приложения «Морской бой», Заключение.

Перечень графического материала:

Листинг программы – 14 страниц

1. Календарный график работы на весь период проектирования: (с указанием

трудоемкости отдельных этапов)

26.02.2019 - 29.03.2019 - Обзор и изучение литературы по теме проекта - 15%;

30.03.2019 - 19.04.2019 - Проектирование приложения – 15%;

20.04.2019 - 13.05.2019 - Реализация приложения – 40%;

14.05.2019 - 17.05.2019 - Тестирование – 15%;

18.05.2019 - 24.05.2019 - Разработка вопросов экономики и охраны труда - 5%;

25.05.2019 - 31.05.2019 - Подготовка пояснительной записки и графической части дипломного проекта – 10%.

Дата выдачи задания « 25 » февраля 20 19г.

Руководитель проекта А.А.Буславский

Задание принял к исполнению: « 25 » февраля 2019 г. Подпись слушателя:

**АННОТАЦИЯ К КУРСОВОМУ ПРОЕКТУ**

## слушателя группы ПОИС 1709в1 Пузанова Кирилла Владимировича

**по специальности «Программное обеспечение информационных систем» на тему: Разработка компьютерной сетевой игры «Морской бой» с помощью технологии Windows Forms платформы .Net Framework**

.NET FRAMEWORK, АРХИТЕКТУРА КЛИЕНТ-СЕРВЕР, ТЕХНОЛОГИЯ ИНТЕЛЛЕКТУАЛЬНОГО КЛИЕНТА WINDOWS FORMS, ИНФОРМАЦИОННАЯ СИСТЕМА, ЯЗЫК С#.

Цель дипломного проектирования - разработка развлекательного программного средства способного обеспечить передачу данных по сети.

Предмет исследования – организация передачи данных по сети.

Областью возможного практического применения разработанного программного средства являются развлекательные интернет – ресурсы, а также приложения иного назначения, использующие подобную архитектуру.

Данный курсовой проект посвящен проектированию и реализации игрового приложения «Морской бой». В нем представлен процесс создания и программирования визуальных библиотечных компонентов технологии Windows Forms, в которых реализованы возможность сетевой игры с помощью библиотечных классов интерфейса для приема и отправки сообщения по сети и применения принципа многопоточности, контроль очередности хода с помощью указателей на методы.

Используя данный проект, можно создать игровое приложение не только для игры по локальной сети Ethernet, но и, модифицировав приложение, реализовать возможность игры через Интернет, а также расширить функционал графического интерфейса.

При выполнении курсового проектирования для разработки интерфейса использовалась среда визуального программирования Microsoft Visual Studio 2013.

Пояснительная записка к курсовому проекту 2 главы и заключение и содержит 42 страницы, в том числе 5 приложений из 14 страниц, 1 таблицу, 9 рисунков, 11 литературных источников.

Автор курсового проекта подтверждает, что приведенный в проекте расчетно-аналитический материал объективно отражает состояние исследуемого процесса (разрабатываемого объекта), все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их авторов.

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ……………………………………………………………....................5](#_bookmark0)

1 [ТИПЫ АРХИТЕКТУР ИНФОРМАЦИОННЫХ СИСТЕМ……………………6](#_bookmark1)

1.1 [Понятие архитектуры системы………………………………………………..6](#_bookmark2)

1.1.1 Архитектура файл-сервер……………………………………………………8

1.1.2. Архитектура клиент-сервер………………………………………………..10

1.1.3 Многоуровневая архитектура…………………………………....................13

1.2 Технология интеллектуального клиента Windows Forms………………….15

1.3 Многопоточность……………………………………………………………..16

1.4 Делегаты……………………………………………………………………….17

2 ПРОЕКТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ С ПОМОЩЬЮ ТЕХНОЛОГИИ WINDOWS FORMS……………………………………………………………………………..19

2.1 Структура игрового приложения «Морской бой»………………………….19

2.1.1 Формирование расстановки кораблей игрока…………………………….20

2.1.2 Реализация взаимодействия между клиентом и сервером……………….21

2.2 Запуск и тестирование работы приложения «Морской бой»………………23

[ЗАКЛЮЧЕНИЕ………………………………………………………....................26](#_bookmark7)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ……………………………..28

[ПРИЛОЖЕНИЕ А Листинг класса Form.cs…………………………………….29](#_bookmark8)

[ПРИЛОЖЕНИЕ Б Листинг класса HomeClient.cs…………………..](#_bookmark9)................35

[ПРИЛОЖЕНИЕ В Листинг класса EnemyServer.cs……………………………37](#_bookmark10)

[ПРИЛОЖЕНИЕ Г Листинг класса Message.cs…………………………………40](#_bookmark11)

ПРИЛОЖЕНИЕ Г Листинг перечисления MessageType.cs.[…………………...42](#_bookmark12)

# ВВЕДЕНИЕ

Проектирование приложений с использованием архитектуры клиент-сервер является распространенным решением задач модульного программирования. Вычислительная модель клиент-сервер является актуальной моделью для распределенных вычислений. С одной стороны, архитектура клиент-сервер представляет собой естественное решение с точки зрения производителя, так как в ней используются все более доступные микрокомпьютеры и сети. С другой стороны, архитектура клиент-сервер, возможно, является идеальным выбором для поддержки выбранного организацией направления бизнеса. Архитектура клиент-сервер создает условия для новых методов организации бизнеса. Компании постоянно пытаются снижать трудовые затраты и избавляться от лишних рабочих мест, что вызвано жесткой рыночной конкуренцией. Архитектура клиент-сервер предоставляет средство автоматизации задач и устранения барьеров для информации, что позволяет компаниям удалять лишние управленческие звенья и увеличивать производительность, не раздувая штаты. Таким образом данная архитектура является решением для широкого спектра практических промышленных проблем и задач.

Цель данного курсового проекта является изучение архитектурного решения клиент-сервер и технологии интеллектуального клиента Windows Forms, отвечающей за графический интерфейс пользователя. Задачами проекта являются проектирование архитектуры клиент-сервер с использованием модели взаимодействия между одновременно выполняемыми программными процессами.

В разделе аналитического обзора литературных источников приводится обзор ряда типовых архитектур информационных систем, описание модели взаимодействия между одновременно выполняемыми процессами, оборудования, интегрированной среды разработки программного обеспечения Microsoft Visual Studio и основные результаты выполненной в процессе проектирования работы.

# ТИПЫ АРХИТЕКТУР ИНФОРМАЦИОННЫХ СИСТЕМ

* 1. **Понятие архитектуры системы**

Основным недостатком персональных компьютеров является их невысокая вычислительная мощность и надежность, а также необходимость в приобретении дополнительных аппаратных средств для устранения изолированности отдельных персональных компьютеров друг от друга. Как правило, пользователям компьютеров нужны и высокая вычислительная мощность и прекрасные свойства персональных компьютеров. Поэтому там, где для выполнения сложных вычислений используются мощные изолированные центральные компьютеры с терминалами, их пользователям периодически приходится ходить на персональные компьютеры для редактирования текстов или выполнения задач, использующих электронные таблицы. Это заставляет пользователей освоить 2 различные операционные системы (на больших машинах обычно установлены OC MVS, VMS, VM, UNIX, а на персональных - MS DOS/MS Windows, OS/2 или Mac) и не решает задачи совместного использования данных.

*Архитектура системы*, согласно стандарту IEEE – это «фундаментальная организационная структура системы, воплощенная в ее компонентах, их взаимоотношениях между собой и с окружением, и принципы, управляющие ее построением и эволюцией» [1].

В настоящее время понятие архитектуры широко используется при анализе, описании, моделировании деятельности организаций (предприятий) как сложных системных объектов. Существование организации (предприятия) предполагает наличие у нее некоторой архитектуры, которая может или нет обеспечить необходимый уровень управления и контроля процессов производства продукции/услуг, добиться соответствия продукции/услуг ожиданиям потребителей, реализовать поставленные цели.

В зависимости от миссии, стратегии развития и долгосрочных бизнес-целей *бизнес-архитектура* определяет необходимые бизнес-процессы, информационные и материальные потоки, поддерживающую их организационную структуру.

*Системная архитектура* определяет совокупность методологических, технологических и технических решений для обеспечения информационной поддержки деятельности организации, определяемой его бизнес-архитектурой, и включает в себя архитектуры приложений, данных и техническую.

*Архитектура приложений* включает прикладные программные системы, поддерживающие выполнение бизнес-процессов, интерфейсы взаимодействия прикладных программных систем между собой и с внешними системами, источниками или потребителями данных, средства и методы разработки и сопровождения приложений.

*Архитектуру данных* определяют базы данных и хранилища данных, системы управления базами данных и хранилищами данных, правила и средства разграничения доступа к данным.

Сетевая архитектура и архитектура платформ представляют *техническую архитектуру*.

*Сетевую архитектуру* образуют вычислительные сети, используемые коммуникационные протоколы, сервисы и системы адресации в сетях, методики обеспечения бесперебойной работы сетей в форс-мажорных условиях.

*Архитектура платформ* включает аппаратные средства вычислительной техники – серверы, рабочие станции, устройства хранения данных и другое компьютерное оборудование, операционные и управляющие системы, утилиты и офисные программные системы, методики обеспечения бесперебойной работы аппаратуры (главным образом, серверов) и баз данных в форс-мажорных обстоятельствах. Архитектура организации является одним из главных средств управления изменениями в бизнесе и технологиях, при этом поддерживает работу менеджеров при анализе потенциальных изменений и их реализации, создает основу для совместной работы бизнес-менеджеров и ИТ-менеджеров, создает единое информационное пространство организации.

*Архитектура информационной системы* – это концептуальное описание структуры, определяющее модель, выполняемые функции и взаимосвязь компонентов информационной системы.

Архитектура информационной системы предусматривает наличие трех компонент:

а) *информационные технологии* – аппаратно-программная компонента, телекоммуникации и данные, совместно обеспечивающие функционирование информационной системы и являющиеся ее главной материальной основой;

б) *функциональные подсистемы* – специализированные программы, обеспечивающие обработку и анализ информации для цельной подготовки документов или принятия решений в конкретной функциональной области на базе информационных технологий;

в) *управление информационными системами* обеспечивает оптимальное взаимодействие информационных технологий, функциональных подсистем и связанных с ними специалистов, а также их развитие в течение всего жизненного цикла информационной системы.

Различают следующие виды архитектур: файл-сервер; клиент-сервер; многоуровневая; архитектура на базе хранилища данных; Internet/Intranet.

В общем случае функции клиентского приложения разбиваются на следующие группы:

а) ввод и отображение данных (презентационная логика) – часть кода клиентского приложения, которая определяет, что пользователь видит на экране, когда работает с приложением. Как правило, получение информации от пользователя происходит посредством различных форм. А выдача результатов запросов – посредством отчетов;

б) бизнес-логика – часть кода клиентского приложения, которая определяет алгоритм решения конкретных задач приложения. Она определяет функциональность и работоспособность системы в целом. Блоки программного кода могут быть распределены по сети и использоваться многократно для создания сложных распределенных приложений;

в) обработка данных внутри приложения (логика базы данных) – часть кода клиентского приложения, которая связывает данные сервера с приложением. Она обеспечивает добавление, модификацию и выборку данных, проверку целостности и непротиворечивости данных и реализацию транзакций.

Физически, функции могут реализовываться одним программным модулем, или же распределяться на несколько параллельных процессов в одном или нескольких узлах сети [2].

В данном проекте рассматривается следующие типы архитектур:

а) файл-серверная архитектура;

б) клиент-серверная архитектура;

в) многоуровневая архитектура.

Архитектура Интернет/Интранет является комплексным объединением многоуровневой архитектуры и технологии Интернет/Интранет, составными частями которой являются Web-сервера для статической и динамической публикации информации и браузеры для просмотра и интерпретации текста. В данном курсовом проекте архитектура Интернет/Интранет не рассматривается.

**1.1.1 Архитектура файл-сервер.** *Файл-сервер* – выделенный сервер, оптимизированный для выполнения файловых операций ввода-вывода и предназначенный для хранения файлов любого типа, обладающий большим объемом дискового пространства. Для повышения надежности хранения данных оборудуется RAID контроллером.

В архитектуре «файл-сервер» сервер выполняет функции хранения данных и кода программы, а клиент – обработку данных. Клиент обращается к серверу на уровне файловых команд, система управления файлами считывает запрашиваемые данные из БД и поблочно передает эти данные клиентскому приложению. Фактически, эта архитектура предполагает автономную работу программного обеспечения ИС на разных компьютерах в сети. Компоненты ИС взаимодействуют только за счет наличия общего хранилища данных под управлением СУБД, поддерживающей файл-серверную архитектуру.

При использовании файл-серверной архитектуры копия СУБД создается для каждого инициированного пользователем сеанса работы с ней, которая выполняется на том же процессоре, что и пользовательский процесс. Вся ответственность за сохранность и целостность базы данных лежит на программе и сетевой операционной системе. Обработка всех данных происходит на рабочих местах, а сервер используется только как разделяемый накопитель. При больших объемах данных и работе во многопользовательском режиме существенно снижается быстродействие.

В архитектуре ИС «файл-сервер» присутствует «толстый» клиент и очень «тонкий» сервер в том смысле, что почти вся работа выполняется на стороне клиента, а от сервера требуется только достаточная емкость дисковой памяти.

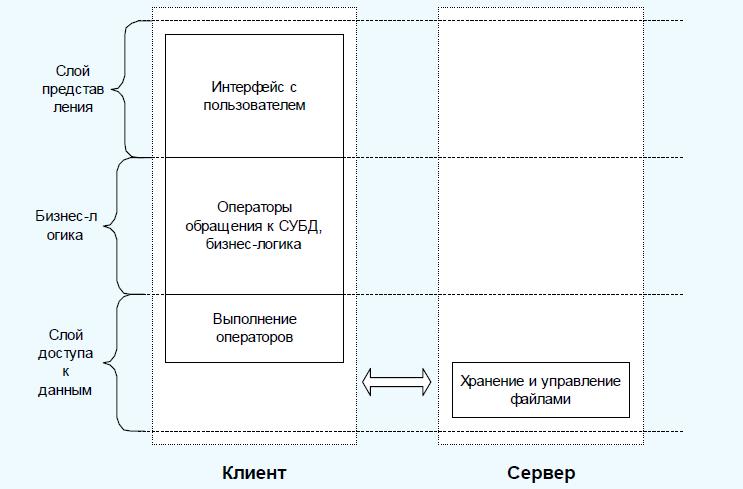
К достоинствам архитектуры «файл-сервер» относят следующие особенности:

а) многопользовательский режим работы с данными;

б) удобство централизованного управления доступом;

в) низкая стоимость разработки.

Недостатки архитектуры с файловым сервером очевидны и вытекают главным образом из того, что данные хранятся в одном месте, а обрабатываются в другом. Это означает, что их нужно передавать по сети, что приводит к очень высоким нагрузкам на сеть и, вследствие этого, резкому снижению производительности приложения при увеличении числа одновременно работающих клиентов. Количество клиентов ограничено десятками. Вторым важным недостатком такой архитектуры является децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным, отсутствие развитых средств защиты данных. Такое решение снижает надежность приложения. На рисунке 1.1 изображена модель файлового сервера [3].



**Рисунок 1.1 – Модель файлового сервера**

**1.1.2. Архитектура клиент-сервер.** Архитектура представляет собой сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам предоставляется доступ к данным.

Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных (физической схемы данных). Теперь клиентские программы манипулируют данными на уровне логической схемы.

Использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных, независимые от аппаратной, а часто и программной части сервера БД и поддерживающие графический интерфейс пользователя (GUI) на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались.

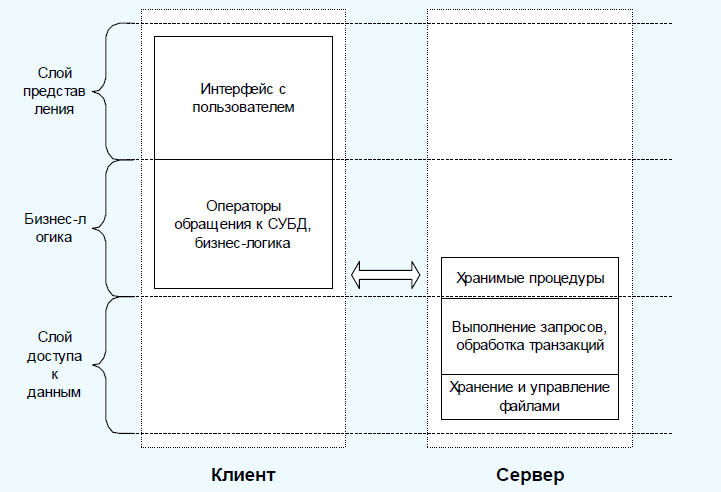
Наиболее часто архитектура клиент-сервер применяется для приложений, созданных с использованием систем управления базами данных (СУБД).

Дальнейшим развитием архитектуры клиент-сервер явилось использование в сети не одного, а нескольких серверов баз данных. Это позволило перейти от работы с локальной БД к работе с распределенной БД.

Причем работа с распределенной базе данных (БД) "прозрачна" для пользователя, т.е. он работает с ней так же, как с локальной БД, не задумываясь о том, на каком сервере лежат его данные. Пользователь обращается к одному из серверов, тот, не найдя у себя нужных данных автоматически обращается к другим серверам.

Многосерверная архитектура сегодня представляется очень перспективной. Она позволяет заменить одну мощную центральную машину на несколько менее мощных и, следовательно, более дешевых, и еще больше распараллелить обработку данных. Кроме того, такая архитектура повышает надежность системы, поскольку при выходе из строя одного из серверов все приложения, работающие с данными других серверов, могут продолжать работу. При выходе из строя части локальной или глобальной сети система может попытаться найти альтернативный путь к нужному серверу (по другим ветвям сети). Кроме того, на локальных серверах могут храниться данные, наиболее часто используемые в данном узле, что позволяет свести к минимуму передачу данных по сети от сервера к серверу.

На рисунке 1.2 представлена модель архитектуры клиент-сервер.



**Рисунок 1.2 – Модель архитектуры клиент-сервер**

Как на клиенте, так и на сервере базовым программным обеспечением является, разумеется, операционная система. Аппаратные платформы и операционные системы клиентов и серверов могут отличаться. В самом деле, в едином окружении могут использоваться разные типы клиентских и серверных платформ и операционных систем.

Однако эти различия не имеют значения, если сервер и клиент используют одни и те же коммуникационные протоколы и поддерживают одинаковые приложения. Взаимодействие клиента и сервера обеспечивается коммуникационным программным обеспечением. Примерами такого программного обеспечения являются набор протоколов TCP/IP, протоколы OSI, а также различные фирменные архитектуры, вроде SNA.

Анализ особенностей архитектуры клиент-сервер представлен в таблице 1.1.

Таблица 1.1 – Достоинства и недостатки клиент-серверной архитектуры

|  |  |
| --- | --- |
| Системная характеристика | Значение |
| Достоинства |  |
| Сеть небольших мощных машин | Если одна машина выйдет из строя, ваша компания все равно сможет продолжать работу |
| Мощные объединения компьютеров | Система предоставляет мощность, позволяющую выполнять работу без монополизации ресурсов. У  конечных пользователей достаточно мощностей для локальной работы |
| Некоторые рабочие станции столь же мощны, как мэйнфреймы, но их стоимость на порядок ниже | Предоставляя вычислительные мощности за меньшие деньги, система позволяет вам тратить сэкономленные средства на другие приобретения или на увеличение ваших доходов |
| Открытые системы | Аппаратуру, программы и услуги можно приобретать у разных поставщиков |
| Легкость наращивания системы | Вашу систему нетрудно модернизировать, как  только ваши потребности изменятся |
| Индивидуальная рабочая среда клиента | Вы можете объединять компьютерные платформы, подбирая их под конкретные нужды подразделений и пользователей |
| Недостатки |  |
| Слабая поддержка | Отдельные части системы не всегда корректно  работают вместе. Бывает довольно трудно найти причину неисправности |
| Недостаток инструментальных  средств обслуживания | При использовании архитектуры клиент-сервер часто приходится искать инструментальные средства на рынке или разрабатывать их самостоятельно |
| Необходимость переобучения | Философия программирования для Маc или  Windows существенно отличается от философии программирования на языке COBOL или С |

Нетрудно заметить, что большинство недостатков классической или двухслойной архитектуры клиент-сервер проистекают от использования клиентской станции в качестве исполнителя бизнес-логики ИС. Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении. При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть [3].

**1.1.3** **Многоуровневая архитектура.** Такие архитектуры более разумно распределяют модули обработки данных, которые в этом случае выполняются на одном или нескольких отдельных серверах. Эти программные модули выполняют функции сервера для интерфейсов с пользователями и клиента - для серверов баз данных. Кроме того, различные серверы приложений могут взаимодействовать между собой для более точного разделения системы на функциональные блоки, выполняющие определенные роли. Например, можно выделить сервер управления персоналом, который будет выполнять все необходимые для управления персоналом функции. Связав с ним отдельную базу данных, можно скрыть от пользователей все детали реализации этого сервера, разрешив им обращаться только к его общедоступным функциям. Кроме того, такую систему очень просто адаптировать к Web, поскольку проще разработать html-формы для доступа пользователей к определенным функциям базы данных, чем ко всем данным.

В трехуровневой архитектуре "тонкий" клиент не перегружен функциями обработки данных, а выполняет свою основную роль системы представления информации, поступающей с сервера приложений. Такой интерфейс можно реализовать с помощью стандартных средств Web-технологии. Это уменьшает объем данных, передаваемых между клиентом и сервером приложений, что позволяет подключать клиентские компьютеры даже по медленным линиям типа телефонных каналов. Кроме того, клиентская часть может быть настолько простой, что в большинстве случаев ее реализуют с помощью универсального браузера. Но если менять ее все-таки придется, то эту процедуру можно осуществить быстро и безболезненно. Трехуровневая архитектура клиент-сервер позволяет более точно назначать полномочия пользователей, так как они получают права доступа не к самой базе данных, а к определенным функциям сервера приложений. Это повышает защищенность системы (по сравнению с обычно архитектурой) не только от умышленного нападения, но и от ошибочных действий персонала.

Для примера рассмотрим систему, различные части которой работают на нескольких удаленных друг от друга серверах. Допустим, что от разработчика поступила новая версия системы, для установки которой в двухуровневой архитектуре необходимо одновременно поменять все системные модули. Если же этого не сделать, то взаимодействие старых клиентов с новыми серверами может привести к непредсказуемым последствиям, так как разработчики обычно не рассчитывают на такое использование системы. В трехуровневой архитектуре ситуация упрощается. Дело в том, что поменяв сервер приложений и сервер хранения данных (это легко сделать одновременно, так как оба они обычно находятся рядом), мы сразу меняем набор доступных сервисов. Таким образом, вероятность ошибки из-за несоответствия версий серверной и клиентской частей резко сокращается. Если в новой версии какой-либо сервис исчез, то элементы интерфейса, обслуживавшие его в старой системе, просто не будут работать. Если же изменился алгоритм работы сервиса, то он будет корректно работать даже со старым интерфейсом.

Многоуровневые клиент-серверные системы достаточно легко можно перевести на Web-технологию - для этого достаточно заменить клиентскую часть универсальным или специализированным браузером, а сервер приложений дополнить Web-сервером и небольшими программами вызова процедур сервера.

Следует отметить и тот факт, что в трехуровневой системе по каналу связи между сервером приложений и базой данных передается достаточно много информации. Однако это не замедляет вычислений, так как для связи указанных элементов можно использовать более скоростные линии. Это потребует минимальных затрат, поскольку оба сервера обычно находятся в одном помещении. Таким образом, увеличивается суммарная производительность системы - над одной задачей теперь работают два различных сервера, а связь между ними можно осуществлять по наиболее скоростным линиям с минимальными затратами средств. Правда, возникает проблема согласованности совместных вычислений, которую призваны решать менеджеры транзакций - новые элементы многоуровневых систем.

Особенностью многоуровневых архитектур является использование менеджеров транзакций (МТ), которые позволяют одному серверу приложений одновременно обмениваться данными с несколькими серверами баз данных. МТ используется для управления распределенными разнородными операциями и согласования действий различных компонентов информационной системы. Следует отметить, что практически любое сложное ПО требует использования менеджера транзакций. Например, банковские системы должны осуществлять различные преобразования представлений документов, т. е. работать одновременно с данными, хранящимися как в базах данных, так и в обычных файлах, - именно эти функции и помогает выполнять МТ [4].

* 1. **Технология интеллектуального клиента Windows Forms**

*Интеллектуальный клиент* — это приложение с полнофункциональным графическим интерфейсом, простое в развертывании и обновлении, способное работать при наличии или отсутствии подключения к Интернету и использующее более безопасный доступ к ресурсам на локальном компьютере по сравнению с традиционными приложениями Windows.

Windows Forms — это технология интеллектуальных клиентов для платформы .NET Framework, набор управляемых библиотек, упрощающих выполнение распространенных задач, таких как чтение и запись в файловой системе. При использовании среды разработки, как Visual Studio, можно создавать приложения интеллектуальных клиентов Windows Forms, которые отображают информацию, запрашивают ввод от пользователей и взаимодействовать с удаленными компьютерами по сети.

В Windows Forms *форма* — это визуальная поверхность, на которой выводится информация для пользователя. Обычно приложение Windows Forms строится путем помещения элементов управления на форму и написания кода для реагирования на действия пользователя, такие как щелчки мыши или нажатия клавиш. *Элемент управления* — это отдельный элемент пользовательского интерфейса, предназначенный для отображения или ввода данных. При выполнении пользователем какого-либо действия с формой или одним из ее элементов управления создается событие. Приложение реагирует на эти события с помощью кода и обрабатывает события при их возникновении. Windows Forms включает широкий набор элементов управления, которые можно добавлять на формы: текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы.

В состав Windows Forms входят многофункциональные элементы пользовательского интерфейса, позволяющие воссоздавать возможности таких сложных приложений, как Microsoft Office. С помощью перетаскивания и вставки *конструктор Windows Forms* в Visual Studio можно легко создавать приложения Windows Forms. Достаточно выделить элемент управления курсором и поместить его в нужное место на форме. Для преодоления трудностей, связанных с выравниванием элементов управления, конструктор предоставляет такие средства, как линии сетки и линии привязки. С помощью Visual Studio или компиляции из командной строки, можно использовать [*FlowLayoutPanel*](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.flowlayoutpanel), [*TableLayoutPanel*](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.tablelayoutpanel) и [*SplitContainer*](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.splitcontainer) элементы управления для создания сложных макетов форм за меньшее время.

Приложение Windows Forms представляет собой [событийно-ориентированное приложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), поддерживаемое Microsoft [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework). В отличие от [пакетных программ](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0), большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в [текстовое поле](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) или клика мышкой по [кнопке](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D0%BE%D0%BF%D0%BA%D0%B0_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)).

На сегодняшний день Windows Forms все еще остается платформой для многих бизнес-приложений, ориентированных на работу с данными. Довольно часто в приложениях можно встретить формы, которые предназначены для ввода или редактирования объектов с большим количеством зависимых свойств. В Windows Forms имеется множество возможностей, которые упрощают и ускоряют реализацию общих задач, таких как создание диалоговых окон, печать, добавление справки и документации, а также локализация приложений на различных языках. Кроме того, в Windows Forms применяется эффективная система безопасности .NET Framework. Благодаря ей можно создавать более надежные приложения [5].

* 1. **Многопоточность**

Одним из ключевых аспектов в современном программировании является многопоточность. Ключевым понятием при работе с многоопоточностью является поток. Поток предствляет некоторую часть кода программы. При выполнении программы каждому потоку выделяется определенный квант времени. И при помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Если у нас, допустим, графическое приложение, которое посылает запрос к какому-нибудь серверу или считывает и обрабатывает огромный файл, то без многопоточности у нас бы блокировался графический интерфейс на время выполнения задачи. А благодаря потокам мы можем выделить отправку запроса или любую другую задачу, которая может долго обрабатываться, в отдельный поток. Поэтому, к примеру, клиент-серверные приложения (и не только они) практически немыслимы без многопоточности.

Основной функционал для использования потоков в приложении сосредоточен в пространстве имен System.Threading. В нем определен класс, представляющий отдельный поток – класс Thread.

Класс Thread определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Основные свойства класса:

а) статическое свойство CurrentContext позволяет получить контекст, в котором выполняется поток;

б) статическое свойство CurrentThread возвращает ссылку на выполняемый поток;

в) свойство IsAlive указывает, работает ли поток в текущий момент;

г) свойство IsBackground указывает, является ли поток фоновым;

д) свойство Name содержит имя потока;

е) свойство Priority хранит приоритет потока - значение перечисления ThreadPriority;

ж) свойство ThreadState возвращает состояние потока - одно из значений перечисления ThreadState;

Некоторые методы класса Thread:

а) статический метод GetDomain возвращает ссылку на домен приложения

б) статический метод GetDomainId возвращает id домена приложения, в котором выполняется текущий поток;

в) статический метод Sleep останавливает поток на определенное количество миллисекунд;

г) метод Abort уведомляет среду CLR о том, что надо прекратить поток, однако прекращение работы потока происходит не сразу, а только тогда, когда это становится возможно. Для проверки завершенности потока следует опрашивать его свойство ThreadState;

д) метод Interrupt прерывает поток на некоторое время;

е) метод Join блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод;

ж) метод Resume возобновляет работу ранее приостановленного потока;

з) метод Start запускает поток;

и) метод Suspend приостанавливает поток [6, 7].

* 1. **Делегаты**

*Делегат* представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, то в итоге получается объект, содержащий ссылку на метод. Более того, метод можно вызывать по этой ссылке. Иными словами, делегат позволяет вызывать метод, на который он ссылается.

По сути, делегат — это безопасный в отношении типов объект, указывающий на другой метод (или, возможно, список методов) приложения, который может быть вызван позднее. В частности, объект делегата поддерживает три важных фрагмента информации:

а) адрес метода, на котором он вызывается;

б) аргументы (если есть) этого метода;

в) возвращаемое значение (если есть) этого метода.

Как только делегат создан и снабжен необходимой информацией, он может динамически вызывать методы, на которые указывает, во время выполнения. Каждый делегат в .NET Framework (включая специальные делегаты) автоматически снабжается способностью вызывать свои методы синхронно или асинхронно. Этот факт значительно упрощает задачи программирования, поскольку позволяет вызывать метод во вторичном потоке выполнения без ручного создания и управления объектом Thread.

Тип делегата объявляется с помощью ключевого слова *delegate*. Ниже приведена общая форма объявления делегата:

delegate возвращаемый\_тип имя (список\_параметров),

где возвращаемый\_тип обозначает тип значения, возвращаемого методами, которые будут вызываться делегатом; имя — конкретное имя делегата; список\_параметров — параметры, необходимые для методов, вызываемых делегатом. Как только будет создан экземпляр делегата, он может вызывать и ссылаться на те методы, возвращаемый тип и параметры которых соответствуют указанным в объявлении делегата.

Самое главное, что делегат может служить для вызова любого метода с соответствующей сигнатурой и возвращаемым типом. Более того, вызываемый метод может быть методом экземпляра, связанным с отдельным объектом, или же статическим методом, связанным с конкретным классом. Значение имеет лишь одно: возвращаемый тип и сигнатура метода должны быть согласованы с теми, которые указаны в объявлении делегата [8, 9].

# ПРОЕКТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ С ПОМОЩЬЮ ТЕХНОЛОГИИ WINDOWS FORMS

* 1. **Структура игрового приложения «Морской бой»**

Приложение «Морской бой», реализованное на языке C# с помощью технологии Windows Form платформы .Net Framework, состоит из следующих частей:

а) логика случайного формирования расстановки кораблей игрока;

б) классы клиента и сервера для работы с протоколом TCP;

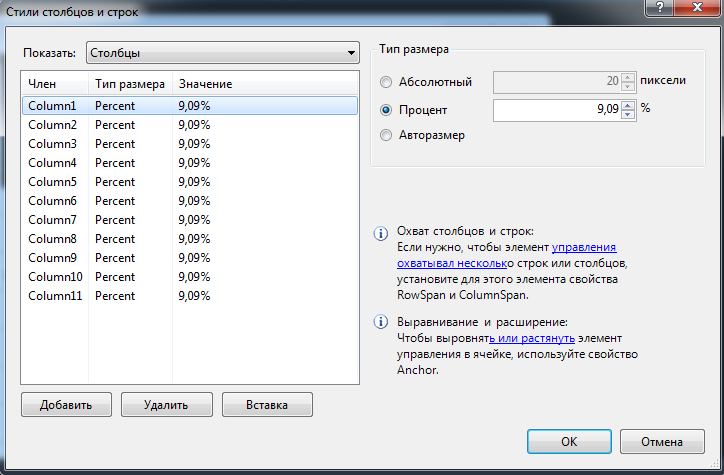
в) класс для формирования сообщения для передачи по сети между клиентской и серверной частями.

Программное средство имеет симметричную клиент-серверную архитектуру, в которой клиентская часть приложения одного игрока является серверной частью приложения другого игрока. Таким образом не возникает необходимости писать отдельный код для клиентской и серверной частей оппонента, что сокращает время разработки и упрощает архитектуру.

Для отображения полей расстановки кораблей игрока и соперника используется компонент панели элементов Windows Forms TableLayoutPanel.

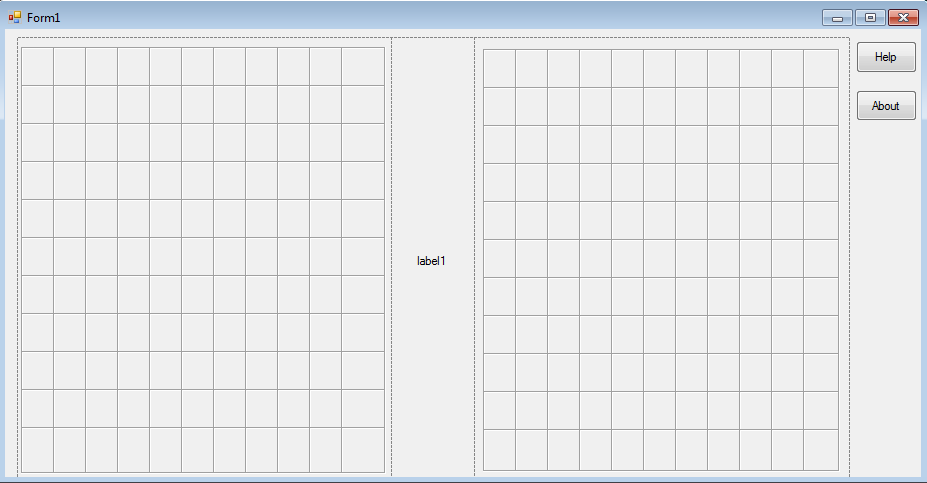
Элемент TableLayoutPanel располагает дочерние элементы управления в виде таблицы, где для каждого элемента имеется своя ячейка. Чтобы установить нужное число строки столбцов таблицы, используются свойства Rows и Columns соответственно.

На рисунке 2.1 представлено разбиение элемента на строки и столбцы с одинаковым размером ячейки сетки.



**Рисунок 2.1 – Формирование сетки размерностью 11 на 11 для элемента TableLayoutPanel**

Непосредственно игровое поле имеет размерность 10 на 10. Верхняя строка и левый столбец из ячеек зарезервированы под нумерацию конкретной ячейки поля. На рисунке 2.2 представлена форма окна приложения в режиме конструктора с результатом формирования сетки для игрового поля обеих игроков.



**Рисунок 2.2 – Форма с элементами TableLayoutPanel для кораблей игрока(слева) и оппонента(справа)**

* + 1. **Формирование расстановки кораблей игрока.** Логика формирования расстановки кораблей игрока реализована с помощью параметра Orientation, параметров X и Y начала координат для формирования корабля в горизонтальной или вертикальной ориентации, массив Ships длин кораблей.

Параметр Orientation типа int отвечает за горизонтально сгенерированный корабль, в случае, если его значение принимает 0, и вертикально сгенерированный корабль, в случае, если значение параметра равно 1. Значения генерируются с помощью объекта класса Math.Random.

Параметры X и Y типа int определяют левый верхний угол начала координат формирования кораблей внутри игрового поля.

Изначально игровое поле заполняется символами Char. После определения ориентации и начала координат случайного формирования кораблей, внутри цикла, перебирающего длины каждого корабля, формируется соответствующий корабль с учетом параметров Orientation и X,Y. Формирование корабля происходит путем замены символа Char, уже содержащегося в данной ячейке, на символ Char, условно обозначающий часть корабля.

Логика формирования кораблей реализована в классе Form.cs непосредственно самой формы, на которой отображаются поля игроков. Код формирования кораблей приведен в приложении А.

* + 1. **Реализация взаимодействия между клиентом и сервером.**

Возможность сетевой игры реализуется посредством двух классов: клиента и сервера, протокола передачи TCP между ними, реализованного внутри библиотечных объектов класса TcpClient и TcpListener, а также объекта Message, который передается между клиентом и сервером. Объект класса HomeClient конструируется с адресом TCP – сервера и номером порта соединения.

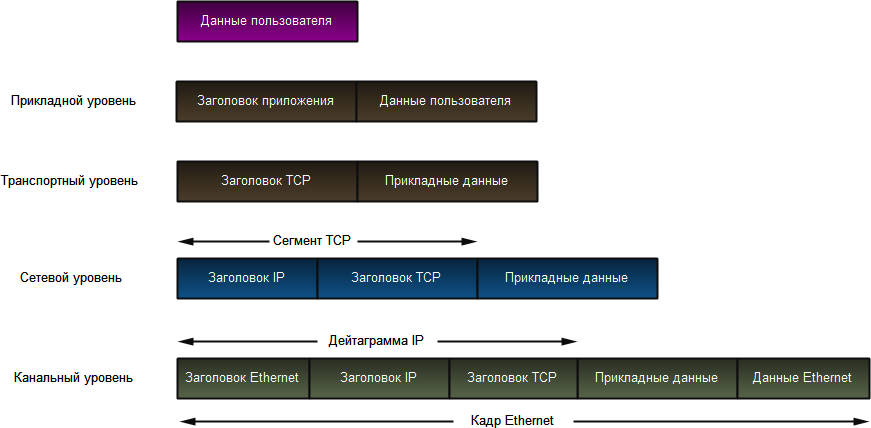
TCP или Transmission Control Protocol, используется как надежный протокол, обеспечивающий взаимодействие через взаимосвязанную сеть компьютеров. TCP проверяет, что данные доставляются по назначению и правильно.

TCP — это ориентированный на соединения протокол, предназначенный для обеспечения надежной передачи данных между процессами, выполняемыми или на одном и том же компьютере или на разных компьютерах. Термин "ориентированный на соединения" означает, что два процесса или приложения прежде чем обмениваться какими-либо данными должны установить TCP-соединение. В этом TCP отличается от протокола UDP, являющегося протоколом "без организации соединения", позволяющим выполнять широковещательную передачу данных неопределенному числу клиентов.

Когда приложение отправляет данные, используя TCP, они перемещаются вниз по стеку протоколов. Данные проходят по всем уровням и в конце концов передаются через сеть как поток битов. Каждый уровень в наборе протоколов TCP/IP добавляет к данным некоторую информацию в форме заголовков. Когда пакет прибывает на конечный узел в сети, он снова проходит через все уровни снизу доверху. Каждый уровень проверяет данные, отделяя от пакета свою информацию в заголовке и наконец данные достигают серверного приложения в той же самой форме, в какой они покинули приложение-клиент.

TCP дает возможность нескольким процессам на одной машине одновременно использовать сокет TCP. Сокет TCP состоит из адреса хоста и уникального номера порта, а TCP-соединение включает два сокета на разных концах сети. Порт может использоваться для нескольких соединений одновременно — один сокет на одном конце может использоваться для нескольких соединений с разными сокетами на другом конце. Примером этой ситуации служит Web-cepвep, слушающий порт 80 и отвечающий на запросы от нескольких компьютеров.

На рисунке 2.3 представлена структура протокола TCP.



**Рисунок 2.3 – Структура протокола TCP**

В отличие от класса Socket, в котором для отправки и получения данных применяется побайтовый подход, классы TcpClient и TcpListener придерживаются потоковой модели. В этих классах все взаимодействие между клиентом и сервером базируется на потоке с использованием класса NetworkStream. Однако при необходимости можно работать с байтами.

Класс TcpClient обеспечивает TCP-сервисы для соединений на стороне клиента. Он построен на классе Socket и обеспечивает TCP-сервисы на более высоком уровне. Класс TcpClient предоставляет простые методы для соединения через сеть с другим приложением сокетов, отправки ему данных и получения данных от него [10, 11].

Для обработки на уровне потока, как канал между двумя соединенными приложениями, используется класс NetworkStream. Прежде чем отправлять и получать любые данные, нужно определить базовый поток. Класс TcpClient предоставляет метод GetStream() исключительно для этих целей. С помощью базового сокета он создает экземпляр класса NetworkStream и возвращает его вызывающей программе. Листинг классов клиентской и серверной части представлен в приложениях Б и В.

Для определения первоначальной очередности хода, а также для отправки координаты ячейки, на которую кликнул мышью игрок и последующей разблокировки формы игрока после «промаха» его оппонента используется объект Message. Сообщение состоит из полей координаты точки Point, идентификационного номера процесса приложения, запущенного у игрока, перечисления MessageType, определяющего тип отправляемых данных: идентификационный номер, координата точки или вызов метода разблокировки формы оппонента после промаха игрока. Листинг классов MessageType и Message представлен в приложениях Г и Д.

Для взаимодействия между игроками путем передачи данных по сети необходимо сохранять состояние объекта в файловом потоке и считывать данные из него. Для этого используется сериализация.

Сериализация представляет процесс преобразования какого-либо объекта в поток байтов. После преобразования мы можем этот поток байтов или записать на диск или сохранить его временно в памяти. А при необходимости можно выполнить обратный процесс - десериализацию, то есть получить из потока байтов ранее сохраненный объект.

Для передачи данных посредством объекта Message, необходимо пометить класс Message атрибутом Serializable. Данный атрибут позволяет преобразовывать объект Message в массив байт для записи или чтения из объекта NetworkStream.

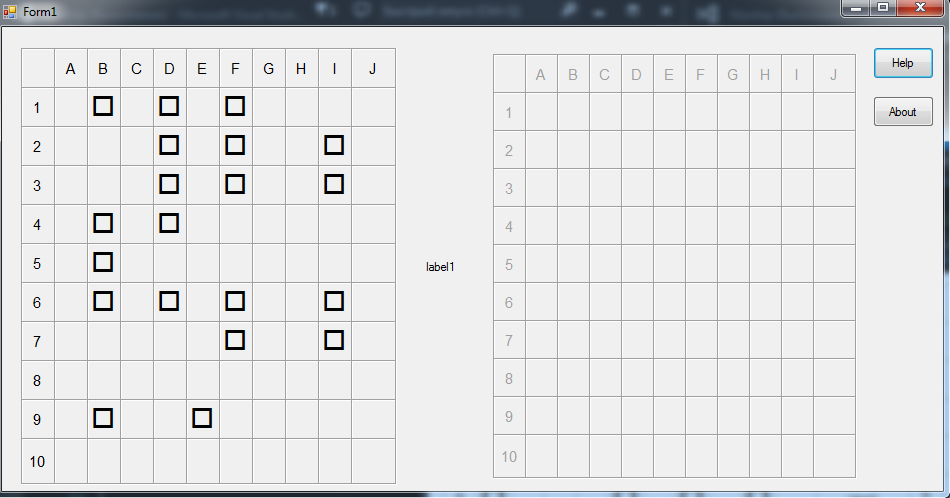
Для сериализации данных в бинарный формат используется объект класса BinaryFormatter.

* 1. **Запуск и тестирование работы приложения «Морской бой»**

Для тестирования работы приложения «Морской бой» на одном компьютере необходимо запустить два различных exe-файла, при этом порты для клиентской части одного игрока и серверной части другого должны быть идентичными.

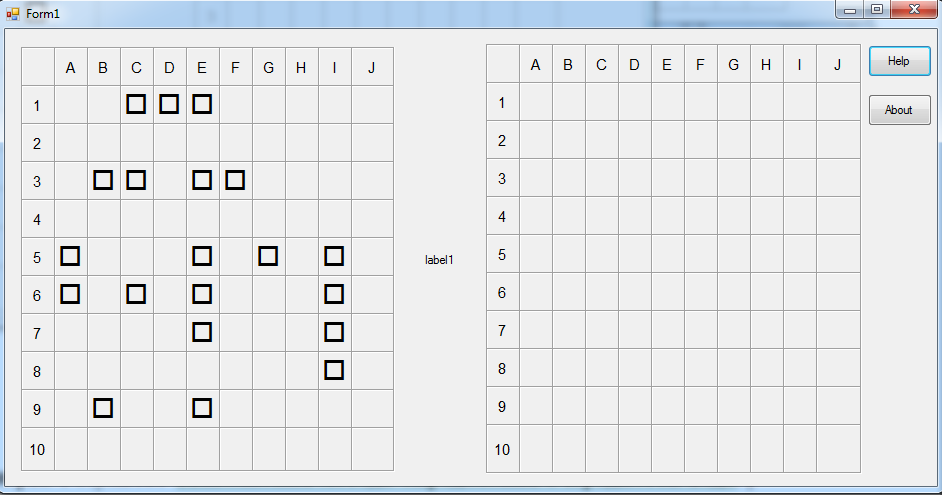
Для тестирования работы приложения на различных компьютерах необходимо наличие подключения по сети Ethernet. При этом порты для клиентской и серверной части игрока должны совпадать с клиентской и серверной частью оппонента.

На рисунках 2.4 – 2.7 представлены скриншот тестирования программы на одном компьютере.

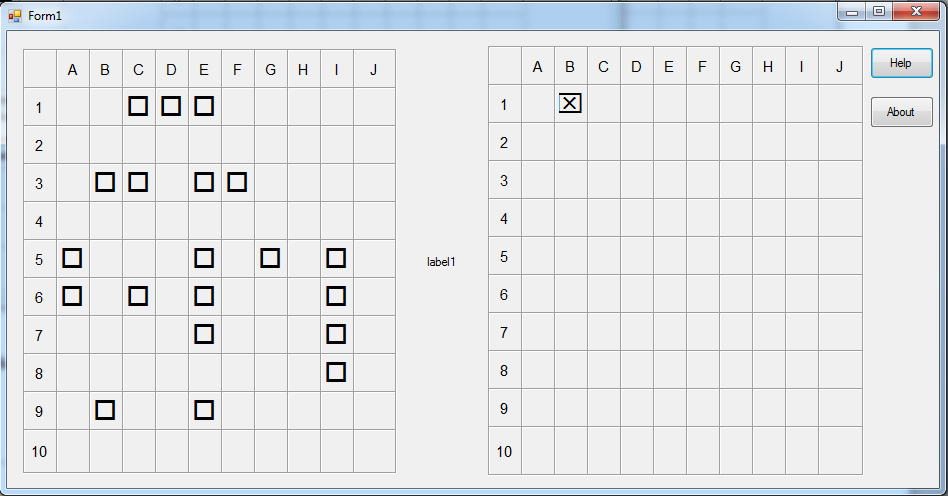


**ЗАКЛЮЧЕНИЕ**

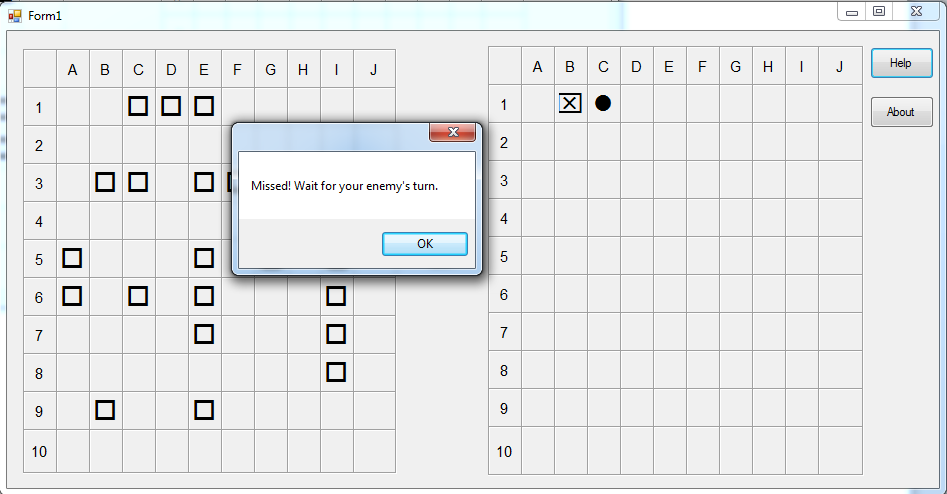
**Рисунок 2.4 – Заблокированная форма одного из игроков в начале игры. Результат выполнения метода определения очередности хода.**

****

**Рисунок 2.5 – Доступная для клика форма другого игрока в начале игры. Результат выполнения метода определения очередности хода.**

****

**Рисунок 2.6 – Результат попадания в корабль соперника. Ход игрока продолжается.**

****

**Рисунок 2.7 – Результат промаха игрока. Ход переходит сопернику.**

Кнопки Help и About справа на окне игрока содержат соответственно описания правил игры «Морской бой» и сведения об авторе приложения.

# ЗАКЛЮЧЕНИЕ

При подготовке курсового проекта были проанализированы теоретические сведения об архитектурах информационных систем и способах реализации низкоуровнего интерфейса передачи данных по сети по протоколу TCP.

В результате курсового проектирования была выполнена задача по разработке игрового приложения «Морской бой» на языке С# с симметричной клиент-серверной архитектурой и классами TcpClient и TcpListener для осуществления передачи данных по сети с использованием технологии реализации графического интерфейса Windows Forms. Модель взаимодействия между двумя одновременно выполняемыми процессами была спроектировано с помощью принципа параллельного выполнения кода – многопоточности.

На этапе реализации логики формирования кораблей была решена задача случайного расположения кораблей на элементе TableLayoutPanel с учетом запрета на их соприконосвение по правилам игры.

На этапе реализации межсетевого взаимодействия между клиентской и серверной частью приложения была решена задача пересылки данных с помощью объекта сериализуемого класса Message. Соответствующая логика поведения серверной части в зависимости от типа пересылаемых данных была реализована с помощью единого принципа формирования сообщения для чтения и отправки с указанием соответствующего типа данных в переменной типа перечисления MessageType. Также была решена задача параллельного выполнения кода на сервере путем запуска серверной части в отдельном потоке.

На этапе определения очередности хода в начале игры была решена задача случайного определения права первого хода путем сравнения идентификационных номеров процессов приложения, запущенных у обоих игроков.

На этапе смены очередности хода в результате промаха была решена задача потокобезопасного обновления состояния формы из другого потока путем вызова метода, который ожидает вызова формы из стороннего потока и выполняет разблокировку формы.

Приложение было испытано в двух режимах: в режиме запуска с одного компьютера и запуска с различных компьютеров. Отличия тестирования приложения заключались в различном принципе определения портов для TCP-сервера. Таким образом была решена задача тестирования и отладки приложения на одном компьютере.

Перспективы использования приложения заключаются в его возможной модификации для игры в сети Интернет, также возможны реализации различных режимов игры: помимо случайно сгенерированных кораблей режим расположения кораблей вручную, реализация оповещения игрока о попадании соперником в его корабль.

Таким образом, поставленные при проектировании приложения задачи были реализованы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] IEEE Recommended practice for architectural description for software-intensive systems. IEEE 1471-2000. – Введ. 09.10.2000. – Institute of Electrical and Electronics Engineers, 2000. – 6 с.

[2] Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов [Электронный ресурс]. – Режим доступа: [<http://www.4stud.info/networking/lecture5.html>](http://belstat.gov.by/) – Дата доступа: 02.06.2019.

[3] Радченко, Г.И. Распределенные вычислительные системы : учеб. пособие / Г.И. Радченко – Челябинск : 2013, 41 – 49 с.

[4] Коржов, В. Многоуровневые системы клиент-сервер / М. Коржов // Сети/Network world – Минск, 1997. – №6, С. 100.

[5] Windows Forms overview [Электронный ресурс]. – Режим доступа[: <https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview>](http://belstat.gov.by/) – Дата доступа: 28.05.2019.

[6] Уроки по C# и платформе .Net Framework [Электронный ресурс]. – Режим доступа[: <https://professorweb.ru/my/csharp/thread_and_files/1/1_1.php>.](http://belstat.gov.by/) – Дата доступа: 1.06.2019.

[7] Программирование на C, C#, Java [Электронный ресурс]. – Режим доступа[: <https://vscode.ru/prog-lessons/potoki-v-si-sharp.html>](http://belstat.gov.by/) – Дата доступа: 2.06.2019.

[8] Шилдт Г. C# 4.0: полное руководство : Научно-популярное издание / Г. Шилдт. – Москва : Издательский дом "Вильямс", 2011. – 473 с.

[9] Троелсен Э. Язык программирования C#5.0 и платформа .NET 4.5 6-е изд. : Научно-популярное издание / Э. Троелсен. – Москва : Издательский дом "Вильямс", 2013. – 652 с.

[10] Полное руководство по языку программирования С# 7.0 и платформе .NET 4.7 [Электронный ресурс]. – Режим доступа[: <https://metanit.com/sharp/tutorial/>](http://belstat.gov.by/) – Дата доступа: 25.05.2019.

[11] Протокол TCP [Электронный ресурс]. – Режим доступа: <https://professorweb.ru/my/csharp/web/level4/4_2.php> – Дата доступа: 1.06.2019.

ПРИЛОЖЕНИЕ А

Листинг класса Form.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Warship

{

public partial class Form1 : Form

{

Random rnd = new Random();

private string Chars = " loxh";

private string Title = " ABCDEFGHIJ";

const byte totalShipsAmount = 20;

int[] Ships = { 4, 3, 3, 2, 2, 2, 1, 1, 1, 1 };

int shipsAmount = 0;

char[,] HomeMap = new char[12, 12];

char[,] EnemyMap = new char[12, 12];

//enable panel for player's turn

public void unlockEnemyShips()

{

if (InvokeRequired)

Invoke(new MethodInvoker(delegate()

{

EnemyShips.Enabled = true;

}));

else

{

EnemyShips.Enabled = true;

}

}

public void GenerateMap(char[,] Map)

{

//Filling array with spaces

for (int i = 0; i < 11; i++)

for (int j = 0; j < 11; j++)

Map[i, j] = Chars[0];

for (int ShipLengthIndex = 0; ShipLengthIndex < Ships.Length; )

{

int ShipLength = Ships[ShipLengthIndex];

//choosing orientation of new ship

int Orientation = rnd.Next(2); //0 - horizontal, 1 - vertical

//choosing left top ship coordinates, keeping inside array

int X = rnd.Next(1, 10 - Orientation \* 3);

int Y = rnd.Next(1, 10 - (1 - Orientation) \* 3);

bool free = true;

for (int i = 0; i < ShipLength; i++)

if (Map[X + i \* Orientation, Y + i \* (1 - Orientation)] != Chars[0])

free = false;

//filling ship cells on the map, moving down or right according to orientation

if (free)

{

for (int i = -1; i < ShipLength + 1; i++)

for (int j = -1; j <= 1; j++)

Map[X + i \* Orientation + j \* (1 - Orientation),

Y + j \* Orientation + i \* (1 - Orientation)] = Chars[4];

for (int i = 0; i < ShipLength; i++)

Map[X + i \* Orientation, Y + i \* (1 - Orientation)] = Chars[2];

ShipLengthIndex++;

}

}

//delete waves

for (int i = 0; i < 11; i++)

for (int j = 1; j < 11; j++)

if (Map[i, j] == Chars[4])

Map[i, j] = Chars[0];

}

//showing map on the form

public void ShowMap(char[,] Map, TableLayoutPanel tlp)

{

for (int i = 1; i < 11; i++)

for (int j = 1; j < 11; j++)

{

Label label = tlp.Controls[i \* 11 + j] as Label;

label.Text = Convert.ToString(Map[i, j]);

}

}

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

HomeShips.Controls.Clear();

for (int i = 0; i < 11; i++)

for (int j = 0; j < 11; j++)

{

Label homeShip = new Label();

homeShip.Dock = DockStyle.Fill;

homeShip.TextAlign = ContentAlignment.MiddleCenter;

homeShip.Font = new Font("Wingdings", 20);

homeShip.Text = Convert.ToString(i) + Convert.ToString(j);

homeShip.Text = " ";

HomeShips.Controls.Add(homeShip, j, i);

Label enemyShip = new Label();

enemyShip.Dock = DockStyle.Fill;

enemyShip.TextAlign = ContentAlignment.MiddleCenter;

enemyShip.Font = new Font("Wingdings", 20);

enemyShip.Text = " ";

enemyShip.Text = Chars.Substring(0, 1);

EnemyShips.Controls.Add(enemyShip, j, i);

enemyShip.Click += Label\_Click\_Enemy;

}

for (int i = 1; i < 11; i++)

{

Label enemyShip = EnemyShips.Controls[i \* 11] as Label;

enemyShip.Font = new Font("Arial", 11);

enemyShip.Text = Convert.ToString(i);

}

for (int j = 1; j < 11; j++)

{

Label enemyShip = EnemyShips.Controls[j] as Label;

enemyShip.Font = new Font("Arial", 11);

enemyShip.Text = Convert.ToString(Title[j]);

}

for (int i = 1; i < 11; i++)

{

Label homeShip = HomeShips.Controls[i \* 11] as Label;

homeShip.Font = new Font("Arial", 11);

homeShip.Text = Convert.ToString(i);

}

for (int j = 1; j < 11; j++)

{

Label homeShip = HomeShips.Controls[j] as Label;

homeShip.Font = new Font("Arial", 11);

homeShip.Text = Convert.ToString(Title[j]);

}

GenerateMap(HomeMap);

ShowMap(HomeMap, HomeShips);

int clientProcessId = Process.GetCurrentProcess().Id;

Message clientProcessIdMessage = new Message(clientProcessId, MessageType.startPlayerMessage);

HomeClient client = new HomeClient();

EnemyServer enemyServer = new EnemyServer(HomeMap, this);

enemyServer.serverStart();

Message serverProcessIdMessage = client.SendAndGetAnswer(clientProcessIdMessage);

if (CompareProcessId(clientProcessIdMessage, serverProcessIdMessage))

{

EnemyShips.Enabled = false;

}

}

private bool CompareProcessId(Message sentMessage, Message receivedMessage)

{

return sentMessage.ProcessId < receivedMessage.ProcessId;

}

private Point GetEnemyCoordinates(Label clickedLabel)

{

var row = EnemyShips.GetRow(clickedLabel);

var column = EnemyShips.GetColumn(clickedLabel);

return new Point(row, column);

}

private void Label\_Click\_Enemy(object sender, EventArgs e)

{

Label a = new Label();

a = sender as Label;

Point enemyCoordinates = GetEnemyCoordinates(a);

HomeClient homeClient = new HomeClient();

Message message = new Message(enemyCoordinates, MessageType.pointMessage);

message = homeClient.SendAndGetAnswer(message);

switch (message.PointValue)

{

case ' ':

a.Text = Convert.ToString(Chars[1]);

a.ForeColor = Color.Black;

MessageBox.Show("Missed! Wait for your enemy's turn.");

EnemyShips.Enabled = false;

HomeClient newHomeClient = new HomeClient();

newHomeClient.SendAndGetAnswer(new Message() { MessageType = MessageType.turnMessage });

break;

case 'o':

if (shipsAmount < totalShipsAmount)

{

a.Text = Convert.ToString(Chars[3]);

a.ForeColor = Color.Black;

shipsAmount++;

}

else

{

MessageBox.Show("Congratulations! You win!");

}

break;

}

}

private void Help\_Click(object sender, EventArgs e)

{

MessageBox.Show("Warship game. You should eliminate all enemy's ships first " +

"by clicking on empty cells to win. Black dot mean you're missed and " +

"must wait for your opponents turn.");

}

private void About\_Click(object sender, EventArgs e)

{

MessageBox.Show("Автор: Пузанов Кирилл Владимирович, группа ПОИС1709в1," +

" кафедра Экономической кибернетики и информатики, " +

"Институт повышения квалификации и переподготовки в области " +

"технологий информатизации и управления” БГУ");

}

}

}

ПРИЛОЖЕНИЕ Б

Листинг класса HomeClient.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net.Sockets;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace Warship

{

public class HomeClient

{

private const int port = 50000;

private const string server = "127.0.0.1";

private TcpClient client;

private Message message;

public HomeClient()

{

client = new TcpClient();

}

private byte[] ObjectToByteArray(object obj)

{

if (obj == null)

return null;

BinaryFormatter bf = new BinaryFormatter();

using (MemoryStream ms = new MemoryStream())

{

bf.Serialize(ms, obj);

return ms.ToArray();

}

}

public Message SendAndGetAnswer (Message message)

{

if (!client.Connected)

{

client.Connect(server, port);

}

NetworkStream stream = client.GetStream();

byte[] data;

IFormatter formatter = new BinaryFormatter();

data = ObjectToByteArray(message);

stream.Write(data, 0, data.Length);

Message receivedMessage = (Message)formatter.Deserialize(stream);

return receivedMessage;

}

}

}

ПРИЛОЖЕНИЕ В

Листинг класса EnemyServer.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net.Sockets;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Warship

{

public class EnemyServer

{

private const int Port = 50001;

private char[,] Map;

private TcpListener server;

private Thread serverThread;

private Form1 mainForm;

public EnemyServer(char[,] HomeMap, Form1 mainForm)

{

Map = HomeMap;

this.mainForm = mainForm;

}

public void serverStart()

{

server = new TcpListener(Port);

serverThread = new Thread(ServerWorker);

serverThread.Start();

}

private byte[] ObjectToByteArray(object obj)

{

if (obj == null)

return null;

BinaryFormatter bf = new BinaryFormatter();

using (MemoryStream ms = new MemoryStream())

{

bf.Serialize(ms, obj);

return ms.ToArray();

}

}

private void ServerWorker()

{

server.Start();

byte[] data;

while (true)

{

Console.WriteLine("Ожидание подключений... ");

// getting incoming connection

TcpClient client = server.AcceptTcpClient();

Console.WriteLine("Подключен клиент. Выполнение запроса...");

// getting network stream for reading and writing

NetworkStream stream = client.GetStream();

IFormatter formatter = new BinaryFormatter();

Message receivedMessage = (Message)formatter.Deserialize(stream);

Message response = new Message();

switch (receivedMessage.MessageType)

{

case MessageType.startPlayerMessage:

response.ProcessId = Process.GetCurrentProcess().Id;

break;

case MessageType.pointMessage:

response.PointValue = Map[receivedMessage.Point.X, receivedMessage.Point.Y];

break;

case MessageType.turnMessage:

mainForm.unlockEnemyShips();

break;

}

// message for client

// convert message to byte array

data = ObjectToByteArray(response);

// sending message

stream.Write(data, 0, data.Length);

Console.WriteLine("Отправлено сообщение: {0}", response);

stream.Close();

client.Close();

}

}

}

}

ПРИЛОЖЕНИЕ Г

Листинг класса Message.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace Warship

{

[Serializable]

public class Message

{

public MessageType MessageType { get; set; }

private int processId;

private Point point;

private int turn;

public char PointValue

{

get;

set;

}

public Message()

{

}

public Message(int processId, MessageType messageType)

{

this.ProcessId = processId;

this.MessageType = messageType;

}

public Message(Point point, MessageType messageType)

{

this.Point = point;

this.MessageType = messageType;

}

public Message(int processId, Point point, int turn)

{

this.ProcessId = processId;

this.Point = point;

this.Turn = turn;

}

public Message(Point point, int turn)

{

this.Point = point;

this.Turn = turn;

}

public Message(int processId, int turn)

{

this.ProcessId = processId;

this.Turn = turn;

}

public Message(int processId)

{

this.ProcessId = processId;

}

public Message(Point point)

{

this.Point = point;

}

public int ProcessId

{

get { return processId; }

set { processId = value; }

}

public Point Point

{

get { return point; }

set { point = value; }

}

public int Turn

{

get { return turn; }

set { turn = value; }

}}}

ПРИЛОЖЕНИЕ Д

Листинг перечисления MessageType.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Warship

{

public enum MessageType

{

startPlayerMessage = 1,

pointMessage = 2,

turnMessage = 3

}

}