Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

СЕТЕВАЯ КОМПЬЮТЕРНАЯ ИГРА «ПОЛЕ ЧУДЕС»

БГУИР КП 6-05-0612-01 008 ПЗ

Студент Еремейко А. А.

Руководитель Болтак С. В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные информационные технологии прочно вошли в сферу развлечений, превратив игровую индустрию в одну из самых динамично развивающихся областей. Компьютерные игры, от захватывающих экшн-приключений до интеллектуальных викторин, стали важной частью досуга миллионов людей, объединяя их вокруг общих интересов и задач. Среди множества игровых жанров особое место занимают викторины, которые сочетают в себе интеллектуальные задачи с развлекательным форматом и духом соревнования.

Одной из самых известных викторин на постсоветском пространстве является телеигра «Поле Чудес», созданная 25 октября 1990 года Владиславом Листьевым на основе американского шоу *«Wheel of Fortune»* [1]. Эта телевизионная программа, ведущим которой на протяжении многих лет является Леонид Якубович, завоевала любовь миллионов зрителей благодаря увлекательной механике: участники угадывают буквы в скрытом слове, вращают барабан для получения очков и соревнуются за призы. Уникальность «Поля Чудес» заключается не только в игровом формате, но и в его культурной значимости, объединяющей поколения.

С развитием компьютерных технологий телевизионные викторины начали обретать цифровую форму. Появление персональных компьютеров и мобильных устройств позволило перенести игровой процесс в виртуальную среду, сделав его доступным широкой аудитории. Однако многие цифровые версии «Поля Чудес» ограничиваются одиночным режимом или локальной игрой на одном устройстве, что не позволяет в полной мере воссоздать дух соревнования и социального взаимодействия, присущий оригинальному шоу. Это создаёт потребность в разработке нового решения, которое сможет объединить игроков в сетевой среде, сохранив при этом аутентичность и увлекательность оригинальной телеигры.

Целью данного курсового проекта является разработка сетевой компьютерной игры «Поле Чудес», которая позволит игрокам соревноваться друг с другом в локальной сети. Программное средство призвано воссоздать классическую механику викторины, дополнив её возможностью многопользовательского взаимодействия, чтобы обеспечить увлекательный и социально ориентированный игровой опыт.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

На сегодняшний день существует множество приложений, созданных по мотивам популярной телевизионной викторины «Поле Чудес». К числу наиболее известных можно отнести «Чудо Поле!», «Поле Чудес Плюс» и «Поле чудес» от *HeroCraft*. Каждое из этих приложений предлагает свой взгляд на классическую игру, а также имеет свои особенности и недостатки. Рассмотрим их более подробно.

«Чудо Поле!» – мобильное приложение для *Android*, созданное для поклонников телевизионной викторины. Игра позволяет игрокам вращать барабан, угадывать буквы и бороться за виртуальные призы в атмосфере, напоминающей телешоу. Одной из ключевых особенностей является возможность выбора словарей разной сложности – с лёгкими словами для новичков и более сложными словами для опытных игроков, что добавляет гибкости игровому процессу. Кроме того, ещё одной особенностью данной игры является наличие мультперсонажей в роли соперников, что добавляет игре легкости и доброжелательности, а также акцентирует внимание на юмористическом подходе к игровому процессу [2]. Однако приложение поддерживает только одиночный режим игры с противниками, управляемыми компьютером, что лишает игру социального взаимодействия. Пользователи также отмечают, что отсутствие функции сохранения прогресса снижает интерес к длительной игре, особенно при многократных прохождениях.

Внешний вид приложения «Чудо Поле!» представлен на рисунке 1.1.

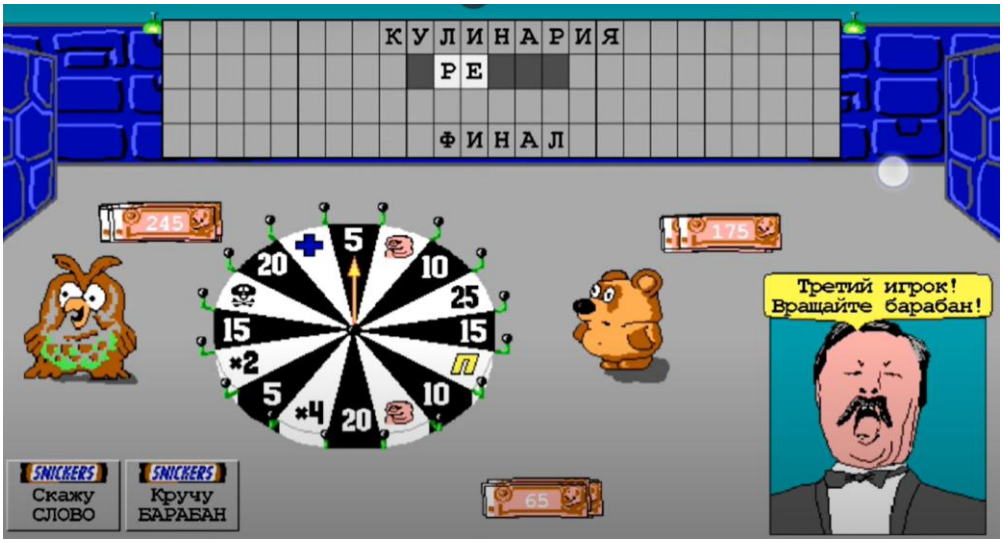

Рисунок 1.1 – Внешний вид приложения «Чудо Поле!»

«Поле Чудес Плюс» – приложение для *Android*, представляющее собой ремейк популярной игры, знакомой многим по кнопочным телефонам [3]. Игра воссоздает атмосферу телешоу с виртуальным ведущим и анимированными участниками. Отличительной чертой является возможность сохранения результатов и призов, что позволяет игрокам отслеживать свои достижения и возвращаться к игре с сохранённым прогрессом. Однако данное приложение ограничивается лишь одиночным режимом игры, что исключает соревнование с реальными игроками. Кроме того, отсутствие настройки уровней сложности для словаря, а также частые рекламные вставки, как отмечают пользователи, делают игровой процесс однообразным и менее комфортным.

На рисунке 1.2 представлен внешний вид приложения «Поле Чудес Плюс».

Рисунок 1.2 – Внешний вид приложения «Поле Чудес Плюс»

«Поле чудес» от *HeroCraft* – компьютерная викторина, выпущенная в 2012 году для ПК [4]. Ключевой особенностью данной игры является наличие режима, который позволяет нескольким игрокам соревноваться на одном устройстве, что идеально подходит для семейных или дружеских посиделок. Кроме того, приложение отличается реалистичным и привлекательным дизайном: виртуальный Леонид Якубович не только внешне похож на реального ведущего, но и использует его знаменитые фразы, а яркая графика усиливает атмосферу телешоу [5]. Однако игра не поддерживает сетевой режим, ограничивая взаимодействие только локальной игрой на одном устройстве, что менее удобно по сравнению с возможностью играть на разных устройствах.

Внешний вид приложения «Поле чудес» от *HeroCraft* представлен на рисунке 1.3.

Рисунок 1.3 – Внешний вид приложения «Поле чудес» от *HeroCraft*

Таким образом, существующие версии игры «Поле Чудес» обладают рядом достоинств, однако их объединяет общий недостаток – отсутствие возможности сетевой игры с реальными игроками в локальной сети. Это существенно ограничивает социальное взаимодействие и снижает соревновательный дух, которые являются важными характеристиками оригинального телешоу. Проведенный анализ подтверждает необходимость в разработке сетевой компьютерной игры «Поле Чудес», которая позволит преодолеть это ограничение, предлагая игрокам более увлекательный и интерактивный опыт совместной игры.

1.2 Постановка задачи

В рамках данного курсового проекта необходимо разработать сетевую компьютерную игру «Поле Чудес», обеспечивающую многопользовательское взаимодействие в локальной сети. Данное программное средство должно гарантировать надёжную передачу данных, удобное управление игровым процессом и поддержку интерактивного соревнования между игроками, воссоздавая атмосферу телевизионной викторины.

Для реализации сетевого взаимодействия будет использована архитектура *P2P (peer-to-peer)*, которая позволит игрокам устанавливать

прямое соединение друг с другом, устраняя необходимость в центральном сервере [6]. Такой подход выбран для повышения автономности системы, снижения затрат на серверную инфраструктуру и упрощения развертывания в локальной сети. Для обнаружения доступных игроков и игровых комнат будет применён протокол *UDP* с широковещательной рассылкой, обеспечивающий высокую скорость передачи данных, что критически важно для быстрого поиска участников [7]. Для подключения к игровой комнате и передачи игровых сообщений будет использован протокол *TCP*, который гарантирует надёжную и последовательную доставку данных, предотвращая потери пакетов и обеспечивая синхронизацию игрового процесса [8].

В процессе разработки будут реализованы функции для управления игроками:

– создание нового игрока;
– выбор существующего игрока для игры;
– удаление игрока;
– просмотр призов, доступных для текущего игрока.

Кроме того, будут реализованы функции для управления игровыми комнатами:

– создание игровой комнаты;
– поиск доступных игровых комнат в локальной сети;
– подключение к существующей комнате для участия в игре;
– выход из игровой комнаты.

Помимо этого, будут реализованы функции, обеспечивающие игровой процесс:

– вращение барабана с обработкой полученного сектора;
– называние буквы с проверкой правильности и начислением очков;
– называние слова целиком с проверкой правильности;
– смена хода игрока;
– отображение хода текущего игрока;
– передача своих действий другим игрокам в игровой комнате;
– определение победителя.

Для разработки программного средства будут использоваться следующие инструменты: *Visual Studio 2022* в качестве интегрированной среды разработки, язык программирования *C#* для реализации логики приложения, а также платформа *Windows Forms* для создания графического интерфейса. Данные об игроках и вопросах будут храниться в файлах формата *JSON*, что обеспечит удобство обработки и высокую читаемость данных [9].

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

При разработке приложения «Поле Чудес» будет использоваться модульная архитектура, включающая 13 основных модулей, каждый из которых отвечает за выполнение определённых функций:

– *Program* – модуль, обеспечивающий запуск приложения и инициализацию данных, включая загрузку информации об игроках и вопросах;

– *MenuForm* – модуль, отображающий главное меню приложения и предоставляющий функционал для перехода к управлению игроками, созданию или поиску игровой комнаты, просмотру призов, и выхода из приложения;

– *RoomForm* – модуль, реализующий интерфейс для создания игровой комнаты, поиска доступных комнат в локальной сети и подключения к ним;

– *GameForm* – модуль, управляющий игровым процессом, включая вращение барабана, угадывание букв или слов, отображение и смену хода игрока, а также определение победителя;

– *PrizesForm* – модуль, отображающий призы, доступные для выбранного игрока;

– *Player* – модуль, хранящий данные об игроке, используемые для локального управления в приложении;

– *PlayerControl* – модуль, управляющий операциями с игроками, такими как создание, выбор, удаление игроков и сохранение данных в файл *players.json*;

– *Question* – модуль, хранящий описание модели вопроса для викторины;

– *QuestionControl* – модуль, отвечающий за загрузку и управление вопросами, включая чтение данных из файла *questions.json*.

– *NetworkManager* – модуль, который обеспечивает сетевое взаимодействие, включая обнаружение игроков с использованием протокола *UDP*, подключение и передачу данных через *TCP*, а также обработку сетевых данных;

– *NetworkData* – модуль, хранящий данные для обмена сетевыми сообщениями между игроками;

– *RoomInfo* – модуль, хранящий информацию о созданной игровой комнате;

– *PlayerInfo* – модуль, хранящий сетевые данные об игроке, необходимые для установления соединения и взаимодействия в игровой комнате.

2.2 Проектирование интерфейса программного средства

Интерфейс программного средства разрабатывается с учётом простоты использования и интуитивной понятности для пользователя. Внешний вид приложения будет минималистичным и функциональным, без избыточных элементов управления.

2.2.1 Главное окно

Главное окно приложения содержит название игры, приветственное сообщение для выбранного игрока, а также кнопки для:

– перехода к окну управления игровыми комнатами;
– открытия панели управления игроками;
– перехода к окну просмотра призов;
– выхода из приложения.

Внешний вид главного окна представлен на рисунке 2.1.

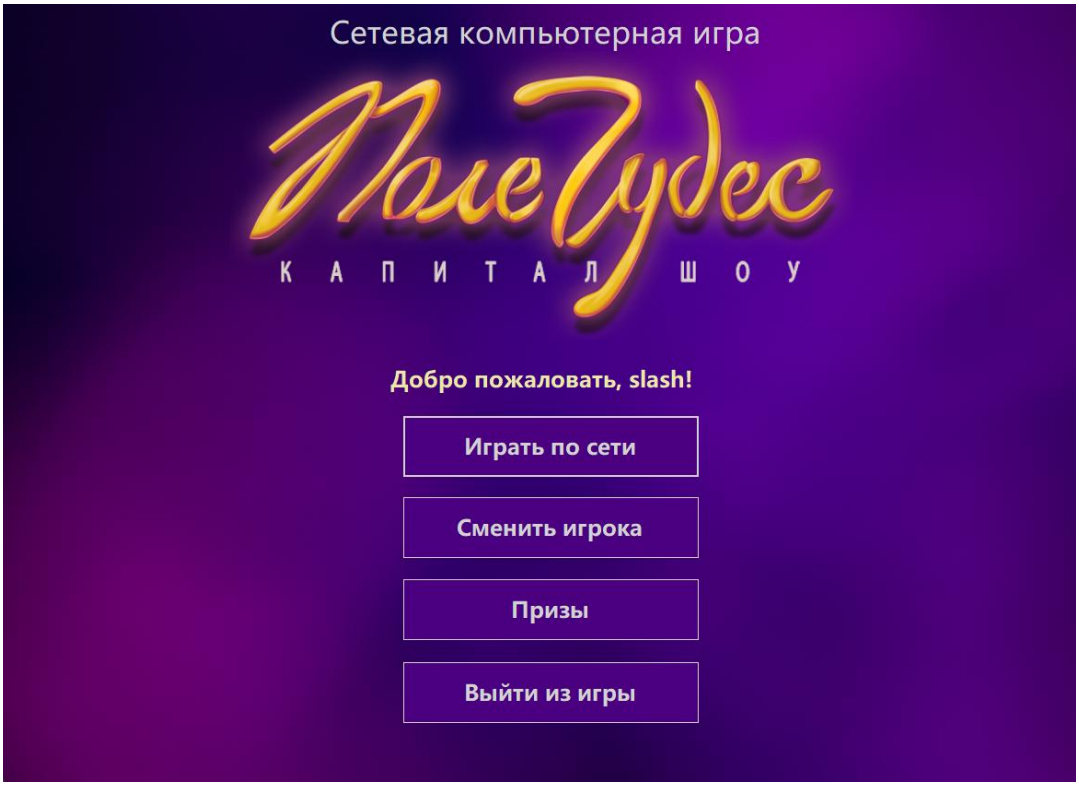

Рисунок 2.1 – Внешний вид главного окна

2.2.2 Панель управления игроками

Панель управления игроками содержит список существующих игроков, а также кнопки для создания игрока, удаления игрока, выбора игрока.

Внешний вид панели управления игроками представлен на рисунке 2.2.

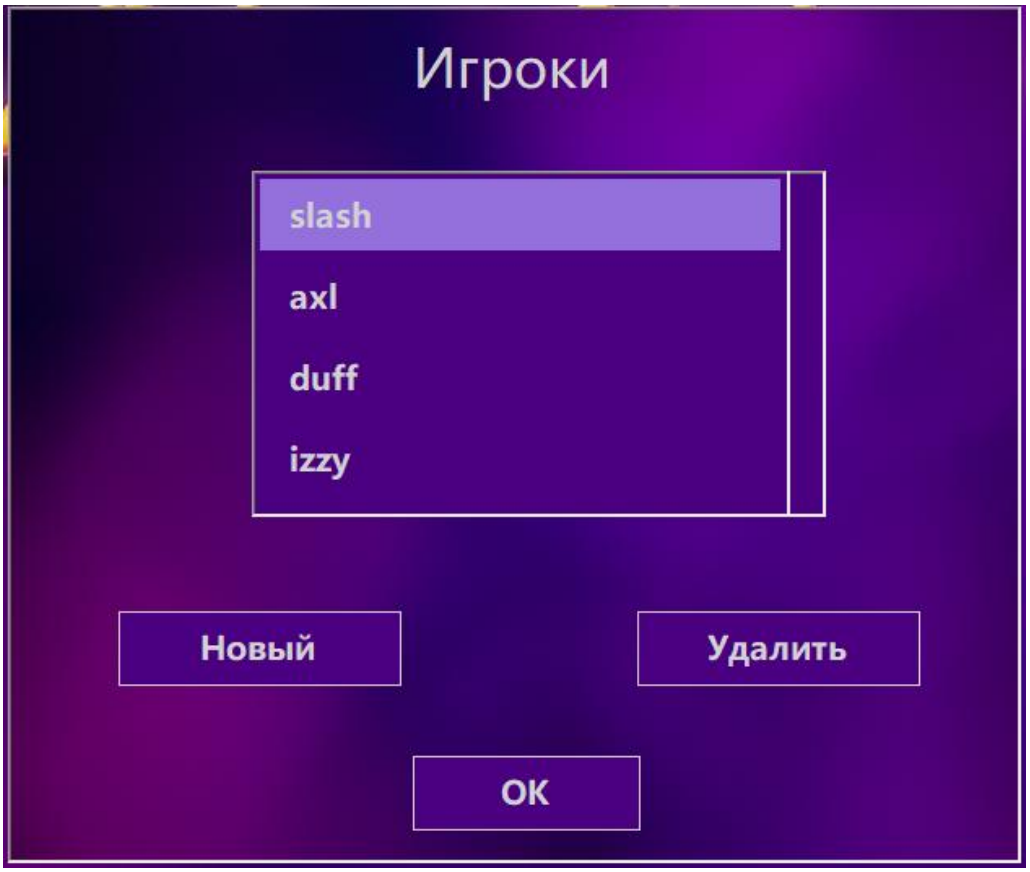

Рисунок 2.2 – Внешний вид панели управления игроками

2.2.3 Окно управления игровыми комнатами

Окно управления игровыми комнатами содержит надпись сетевого адреса текущего игрока и кнопки для:

– перехода в главное меню;
– открытия панели создания игровой комнаты;
– открытия панели поиска доступных игровых комнат.

Внешний вид окна управления игровыми комнатами представлен на рисунке 2.3.

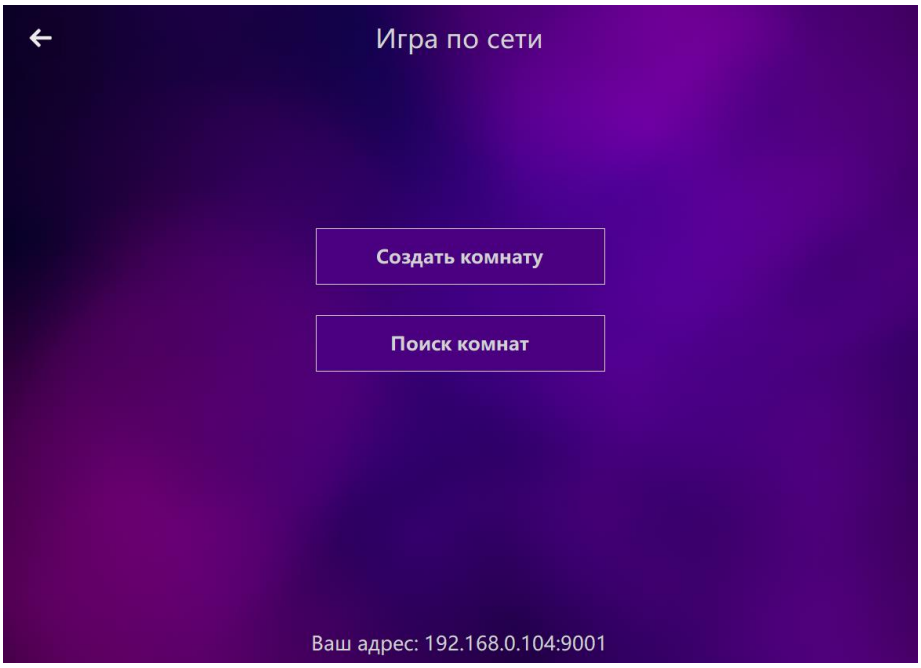

Рисунок 2.3 – Внешний вид окна управления игровыми комнатами

2.2.4 Панель создания игровой комнаты

Панель создания игровой комнаты содержит текстовое поле для ввода названия комнаты, а также кнопки создания игровой комнаты, и отмены создания игровой комнаты.

На рисунке 2.4 представлен внешний вид панели создания игровой комнаты.

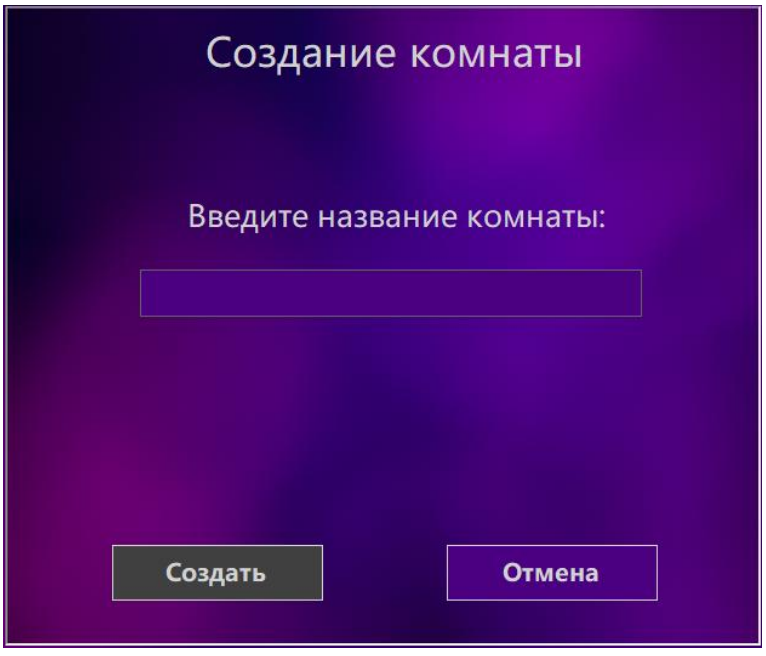

Рисунок 2.4 – Внешний вид панели создания игровой комнаты

2.2.5 Панель поиска доступных игровых комнат

Панель поиска доступных игровых комнат содержит список игровых комнат, найденных в локальной сети, с информацией о комнате, которая включает в себя название комнаты, количество игроков, и информацию об игроках – их имя и сетевой адрес, а также значок лидера для создателя комнаты. Кроме того, на панели поиска доступных игровых комнат имеются кнопки подключения к выбранной комнате и перехода в окно управления игровыми комнатами.

Внешний вид панели поиска доступных игровых комнат представлен на рисунке 2.5.

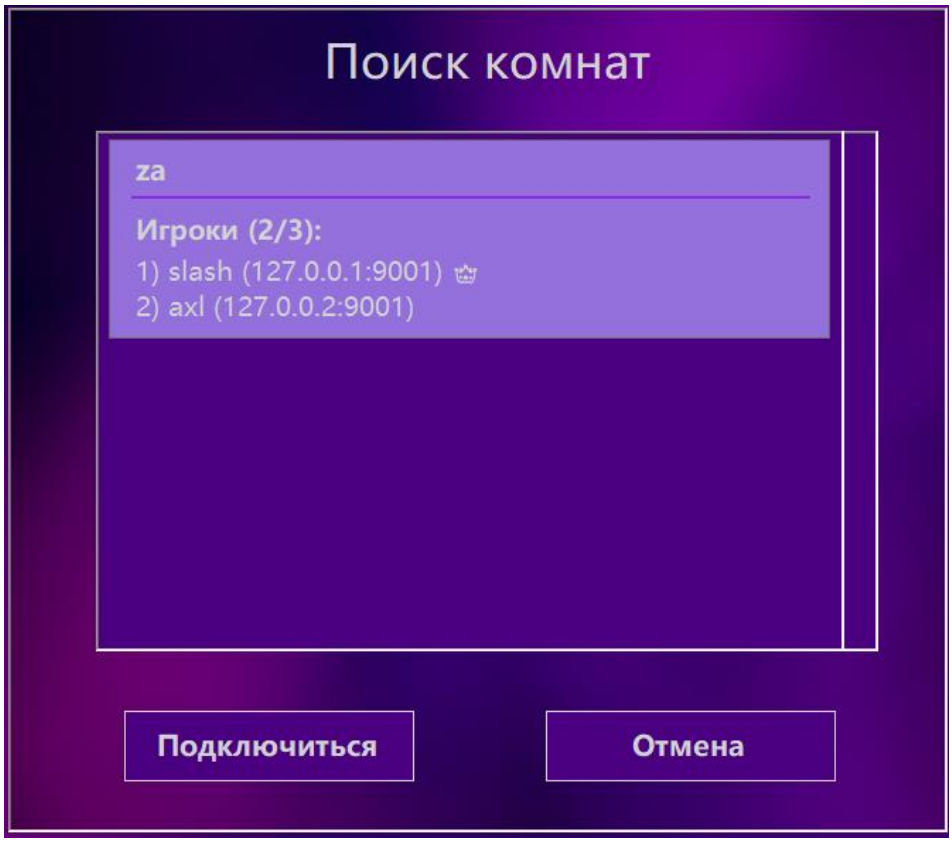

Рисунок 2.5 – Внешний вид панели поиска доступных игровых комнат

2.2.6 Окно просмотра призов

Окно просмотра призов содержит информацию о призах, имеющихся у выбранного игрока – названия и картинки призов, а также их количество.

Внешний вид окна просмотра призов представлен на рисунке 2.6.

Рисунок 2.6 – Внешний вид окна просмотра призов

2.2.7 Окно игрового процесса

Окно игрового процесса содержит кнопки «Крутить барабан» и «Назвать слово», которые доступны только для игрока, который сейчас ходит. Также окно игрового процесса содержит табло, на котором представлены ячейки букв загаданного слова и тема игры. Кроме того, в окне имеется картинка барабана и стрелка, указывающая на текущий сектор, а также картинка ведущего и картинка облака, в котором отображается речь ведущего, чтобы игроки знали, что происходит в данный момент игры. Помимо этого, в данном окне содержится информация об игроках – имя, сетевой адрес и заработанное количество очков.

Рисунок 2.7 – Внешний вид окна игрового процесса

2.3 Проектирование функционала программного средства

Сетевое взаимодействие приложения «Поле Чудес» основано на обнаружении игровых комнат через широковещательные *UDP*-пакеты и установлении надёжных соединений между игроками с использованием протокола *TCP*. Для обеспечения сетевой функциональности приложение должно реализовывать четыре ключевые функции, которые обеспечивают логически связанный процесс взаимодействия: отправка *UDP*-рассылки, приём *UDP*-пакетов, инициирование *TCP*-подключения и принятие *TCP*-соединений.

2.3.1 Отправка *UDP*-рассылки

Лидер игровой комнаты периодически отправляет широковещательные *UDP*-пакеты, чтобы сообщить другим устройствам в локальной сети о существовании комнаты. Модуль *NetworkManager* формирует сообщение, содержащее имя комнаты, сетевой адрес, порт, количество игроков, их данные и информацию о загаданном вопросе. Это сообщение отправляется на широковещательный адрес, после чего процесс повторяется через заданный интервал времени, чтобы поддерживать актуальность информации о комнате. Блок-схема метода отправки *UDP*-рассылки представлена на рисунке 2.8.

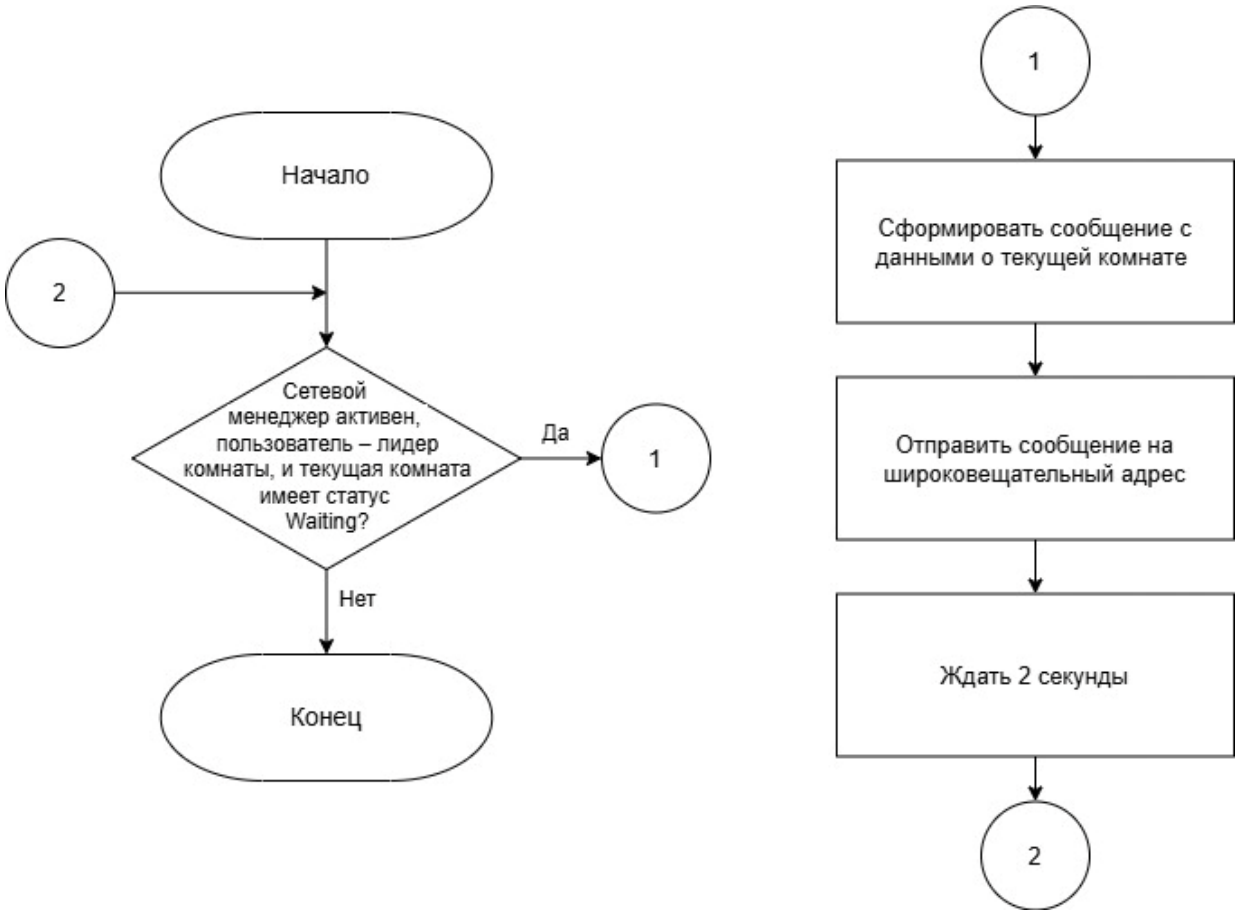

Рисунок 2.8 – Блок-схема метода отправки *UDP*-рассылки

2.3.2 Приём *UDP*-пакетов

Для обнаружения доступных игровых комнат приложение принимает *UDP*-пакеты от других игроков. Модуль *NetworkManager* прослушивает входящие сообщения на заданном порту. Полученное сообщение анализируется для извлечения данных о комнате, включая её имя, сетевой адрес, порт, количество игроков, статус и вопрос. Если комната новая, она добавляется в список доступных комнат, а если уже существует, то её данные обновляются. Список комнат отображается в интерфейсе пользователя. Блок-схема метода приёма *UDP*-пакетов представлена на рисунке 2.9.

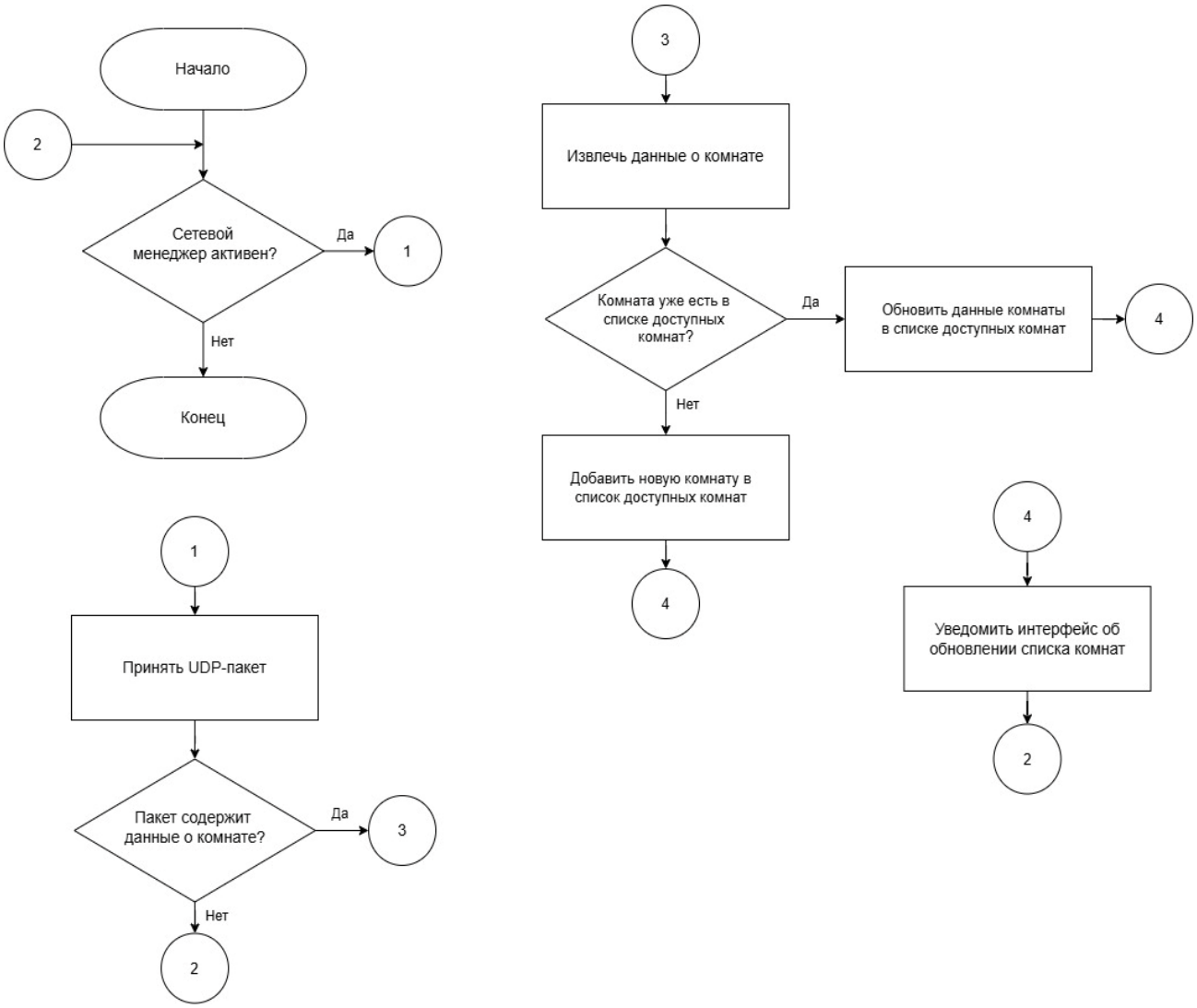

Рисунок 2.9 – Блок-схема метода приёма *UDP*-пакетов

2.3.3 Инициирование *TCP*-подключения

Подключение к игровой комнате начинается с выбора комнаты из списка и нажатия кнопки «Подключиться». Модуль *NetworkManager* проверяет, не заполнена ли комната, и создаёт локальную информацию о комнате на основе полученных данных. Затем для каждого игрока в комнате, кроме самого подключающегося, устанавливается *TCP*-соединение через создание сокета и подключение к указанному *IP*-адресу и порту. После подключения отправляется сообщение с информацией о новом игроке, и запускается процесс обмена данными. Блок-схема метода инициирования *TCP*-подключения представлена на рисунке 2.10.

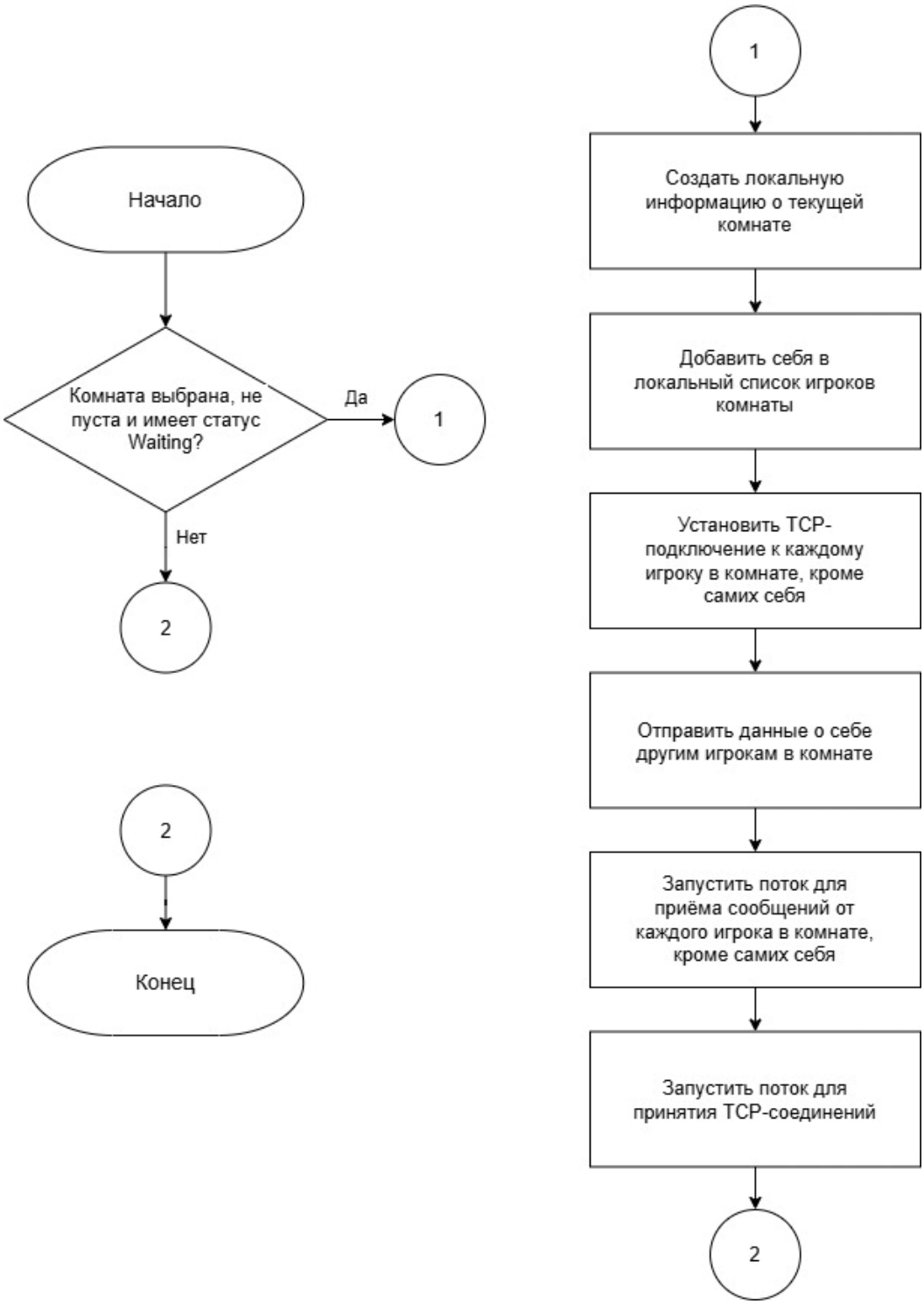

Рисунок 2.10 – Блок-схема метода инициирования *TCP*-подключения

2.3.4 Принятие *TCP*-соединений

Для принятия входящих соединений от других игроков модуль *NetworkManager* использует *TCP*-сервер, прослушивающий входящие подключения. При поступлении нового соединения создаётся сокет для клиента, и запускается отдельный поток для обработки сообщений от этого игрока. Это позволяет одновременно поддерживать несколько соединений,

обеспечивая взаимодействие между игроками в комнате. Блок-схема метода принятия *TCP*-соединений представлена на рисунке 2.11.

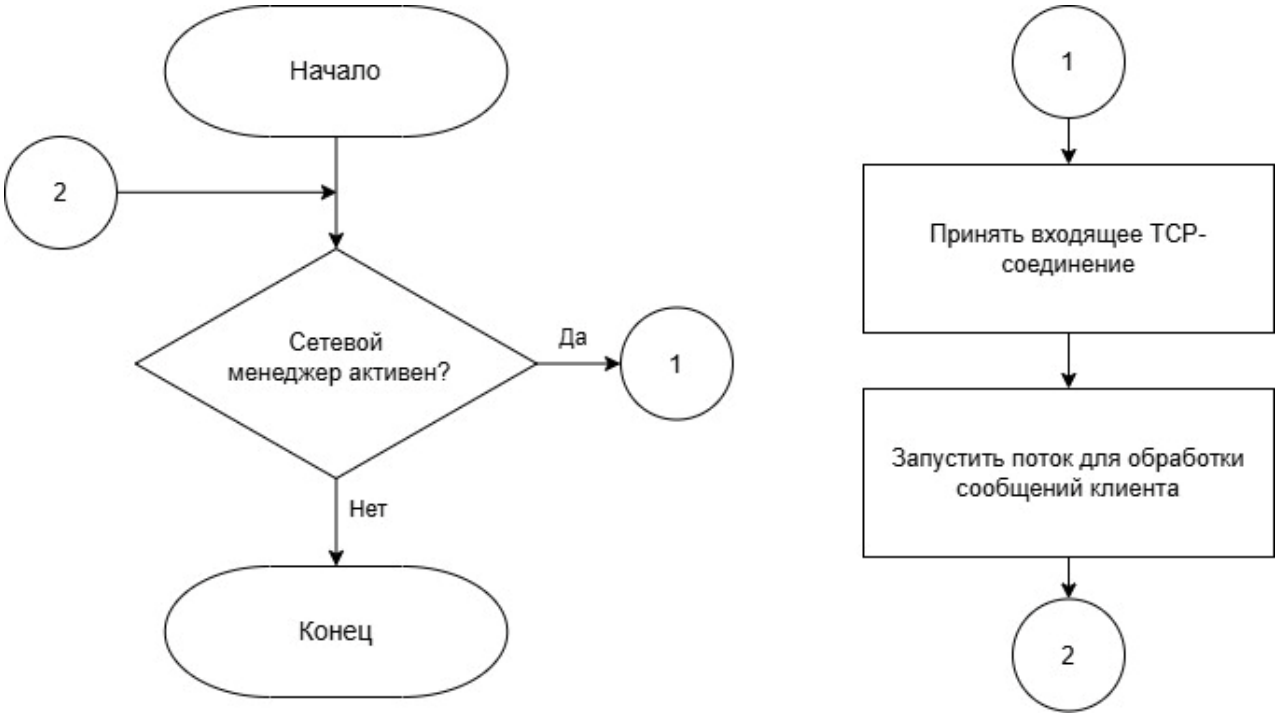

Рисунок 2.11 – Блок-схема метода принятия *TCP*-соединений

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Сетевое взаимодействие

Сетевое взаимодействие является основой многопользовательского режима приложения и реализовано через модуль *NetworkManager*, который управляет обнаружением игровых комнат, установлением соединений и передачей данных между игроками. Использование протоколов *UDP* и *TCP* обеспечивает баланс между скоростью обнаружения и надёжностью передачи игровых событий, таких как вращение барабана или выбор буквы.

Обнаружение игровых комнат осуществляется с помощью широковещательных *UDP*-пакетов, отправляемых лидером комнаты через метод *SendUdpBroadcast*. Лидер периодически формирует сообщение, содержащее информацию о комнате: её название, сетевой адрес, *TCP*-порт, количество игроков, их имена, *IP*-адреса, порты и текущий игровой вопрос, выбранный через *QuestionControl*. Это сообщение отправляется на широковещательный адрес локальной сети, что позволяет другим устройствам обнаружить комнату. Для предотвращения перегрузки сети отправка происходит с интервалом в две секунды. На стороне клиента модуль *NetworkManager* прослушивает входящие *UDP*-пакеты через метод *ReceiveUdpBroadcasts*. Полученные сообщения анализируются для извлечения данных о комнате, после чего информация либо добавляется в список доступных комнат, отображаемый в форме *RoomForm*, либо обновляет существующую запись, если комната уже известна. Событие *RoomsUpdated* уведомляет интерфейс об изменениях, обеспечивая актуальность списка.

Подключение к комнате инициируется пользователем через выбор комнаты в интерфейсе *RoomForm* и нажатие кнопки «Подключиться», что активирует метод *JoinRoom*. Модуль *NetworkManager* проверяет, не заполнена ли комната (максимум три игрока), и создаёт локальную копию информации о комнате, включая список игроков и текущий вопрос. Затем устанавливаются *TCP*-соединения с каждым игроком в комнате, кроме самого подключающегося. Для этого создаются сокеты, которые подключаются к *IP*-адресам и портам других игроков. После успешного соединения отправляется сообщение с данными нового игрока (имя, *IP*, порт), что позволяет другим участникам обновить свои списки. Событие *PlayerJoined* уведомляет интерфейс *GameForm* о новом игроке, обновляя отображение участников. Одновременно запускается процесс принятия входящих *TCP*-соединений через метод *AcceptTcpClients*, который создаёт сервер, прослушивающий

входящие запросы. При поступлении соединения создаётся отдельный поток для обработки сообщений, что обеспечивает асинхронное взаимодействие.

Передача игровых событий, таких как результат вращения барабана или выбор буквы, осуществляется через *TCP* с использованием методов *SendSpinResul*t и *SendSelectedLetter*. Сообщения формируются с заголовком, включающим тип события и длину данных, и отправляются всем игрокам в комнате. Входящие сообщения обрабатываются через метод *ReceiveMessages*, который анализирует тип сообщения и вызывает соответствующие события, такие как *SpinBarabanReceived* или *SelectedLetterReceived*, для обновления игрового состояния. Для обеспечения надёжности предусмотрена обработка сетевых ошибок: при сбоях сокеты закрываются, а интерфейс уведомляется о возможных проблемах, что минимизирует сбои в игре.

3.2 Управление игровым процессом

Игровой процесс реализован в модуле *GameForm*, который отвечает за визуализацию викторины, управление ходами игроков и синхронизацию игровых действий. Основные функции включают вращение барабана, угадывание букв и слов, а также отображение текущего состояния игры, таких как тема, загаданное слово и очки игроков. Интерфейс формы включает элементы управления, такие как кнопки «Крутить барабан» и «Назвать слово», панель с буквами, табло с загаданным словом и область для сообщений ведущего, что создаёт атмосферу телевизионной викторины.

Инициализация игрового процесса начинается с отображения информации о комнате, включая её название и список игроков, отсортированных по времени подключения, через метод *JoinRoom* в *GameForm*. Интерфейс обновляется динамически при присоединении новых игроков благодаря событию *PlayerJoined*, которое вызывает обновление меток с именами и адресами участников. Когда комната заполняется тремя игроками, запускается обратный отсчёт, реализованный через асинхронные задержки, во время которого ведущий (элемент *lblMessageYakub*) отображает сообщения, такие как «До начала игры осталось – 5». После отсчёта отображается приветственное сообщение и тема игры, выбранная из файла *questions.json* через *QuestionControl*. Загаданное слово представлено на табло в виде панелей, каждая из которых скрывает букву, что реализовано через динамическое создание элементов в *panelWord*.

Вращение барабана инициируется нажатием кнопки «Крутить барабан», доступной только текущему игроку, что определяется через метод *DisplayCurrentPlayerTurn*. Модуль *GameForm* выбирает случайный сектор (например, «300», «ПЛЮС», «Банкрот») и запускает анимацию, которая

имитирует вращение барабана путём смены изображений в *pictureBoxBaraban*. Анимация включает два полных оборота и замедление в конце, создавая реалистичный эффект. Результат вращения отправляется другим игрокам через событие *SpinBarabanReceived*, вызываемое методом *SendSpinResult*. Сектор обрабатывается в зависимости от его типа: числовые сектора (100–1000) предлагают назвать букву, «x2» или «x3» умножают очки, «ПЛЮС» позволяет открыть любую букву, «ПРИЗ» предлагает выбор между призом и игрой, а «Банкрот» или «0» передают ход следующему игроку. Обработка сектора сопровождается сообщениями ведущего, отображаемыми в *lblMessageYakub*.

Угадывание букв реализовано через взаимодействие с панелью букв (*panelLetters*) или выбором позиции для сектора «ПЛЮС». Игрок выбирает букву, кликая по метке, что вызывает событие *LetterLabel_Click*. Выбранная буква отправляется другим игрокам через метод *SendSelectedLetter*, и обрабатывается локально через метод *ProcessLetter*. Модуль проверяет, содержится ли буква в загаданном слове, и, если да, открывает все её вхождения на табло, начисляя очки в зависимости от сектора (например, 300 очков за каждую букву или умножение при «x2»). Если буква неверна, ход передаётся следующему игроку через метод *SwitchPlayerTurn*. Событие *SelectedLetterReceived* синхронизирует изменения на всех устройствах, обновляя табло и очки. Угадывание слова целиком возможно через кнопку «Назвать слово», которая проверяет введённый текст и завершает игру при правильном ответе.

Синхронизация игрового состояния обеспечивается через *TCP*-соединения, управляемые *NetworkManager*. Все действия, такие как вращение барабана или выбор буквы, немедленно передаются другим игрокам, а входящие события обновляют интерфейс *GameForm*. Для предотвращения визуальных артефактов применены методы двойной буферизации, что устраняет мерцание при обновлении элементов, таких как *panelCloudYakub* или *panelLetters*. Переключение ходов между игроками реализовано через отслеживание текущего игрока по индексу, основанному на времени подключения, что гарантирует справедливую очерёдность.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование сетевой игры «Поле Чудес» проводилось с целью выявления и устранения ошибок, обеспечения стабильности работы приложения и корректности взаимодействия пользователей в многопользовательском режиме. В процессе тестирования были проверены как клиентская часть, реализованная на платформе *Windows Forms*, так и сетевая логика, основанная на *P2P*-архитектуре с использованием протоколов *UDP* и *TCP*. Выявленные проблемы касались сетевой синхронизации, обработки пользовательских действий и работы интерфейса. Все обнаруженные недочёты были устранены, что позволило добиться устойчивой работы программы.

Одной из ключевых проблем, выявленных на этапе тестирования, была некорректная синхронизация обновления интерфейса при получении сетевых сообщений. При обработке события *SelectedLetterReceived*, вызываемого в *GameForm* для обновления табло (*panelWord*) и сообщений ведущего (*lblMessageYakub*), возникало исключение, связанное с доступом к элементам управления из потока, отличного от основного *UI*-потока. Это приводило к сбоям в отображении букв на табло у некоторых игроков. Для решения проблемы была реализована проверка необходимости вызова метода в основном потоке с использованием свойства *InvokeRequired* [10]. Если обновление инициировалось из сетевого потока, метод делегировался в основной поток через *Invoke*. Данная проверка устранила исключения и обеспечила корректное обновление интерфейса при сетевых операциях. Пример реализации:

```
if (InvokeRequired) {
   Invoke (new Action<string>(ProcessLetter), selectedLetter);
   return;
}
```

Большинство проблем возникало на этапе реализации сетевых функций и было устранено в процессе модульного и ручного тестирования. Финальная версия программного средства обеспечивает стабильную работу, надёжную обработку исключений и корректное взаимодействие пользователей в сетевой среде. Приложение успешно воспроизводит игровой процесс телевизионной викторины, поддерживая синхронизацию действий и интуитивное управление.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Интерфейс программного средства

Интерфейс приложения состоит из набора окон и панелей, каждое из которых выполняет определённую функцию: управление игроками, создание и поиск игровых комнат, игровой процесс и просмотр призов. Все элементы управления имеют минималистичный дизайн, что упрощает навигацию и сосредотачивает внимание пользователя на игровом процессе.

5.1.1 Главное окно и панель управления игроками

Главное окно, реализованное в модуле *MenuForm*, отображается при запуске приложения. Оно содержит название игры, приветственное сообщение с именем выбранного игрока и четыре основные кнопки: переход к окну управления игровыми комнатами, открытие панели управления игроками, переход к окну просмотра призов и выход из приложения. Панель управления игроками открывается по нажатию соответствующей кнопки и включает список существующих игроков, а также кнопки для создания, выбора или удаления игрока. Внешний вид главного окна представлен на рисунке 5.1, а панель управления игроками – на рисунке 5.2.

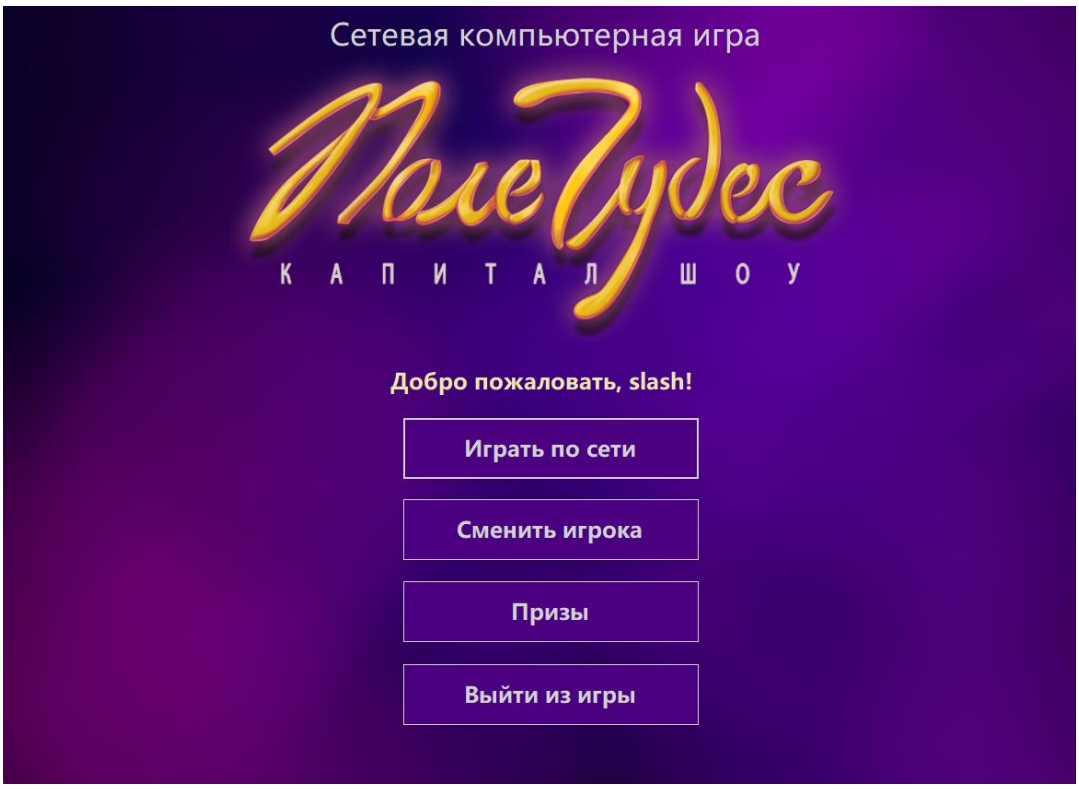

Рисунок 5.1 – Главное окно приложения

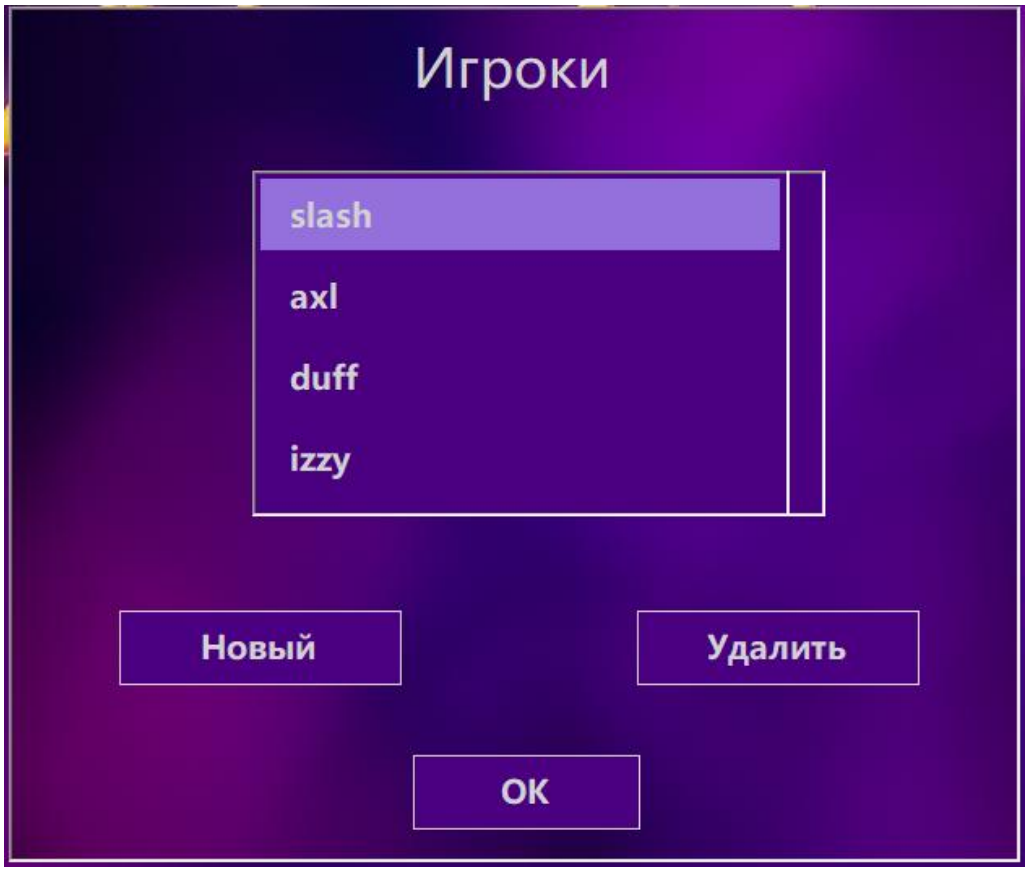

Рисунок 5.2 – Панель управления игроками

5.1.2 Окно управления игровыми комнатами и связанные панели

Окно управления игровыми комнатами, реализованное в модуле *RoomForm*, отображает сетевой адрес текущего игрока и предоставляет кнопки для возврата в главное меню, открытия панели создания комнаты и открытия панели поиска доступных комнат. Панель создания комнаты содержит поле для ввода названия комнаты и кнопки для создания или отмены действия. Панель поиска комнат показывает список доступных игровых комнат с информацией о названии, количестве игроков, их именах, сетевых адресах и значке лидера для создателя комнаты, а также кнопки для подключения к выбранной комнате или возврата в окно управления комнатами. Внешний вид окна управления комнатами представлен на рисунке 5.3, панели создания комнаты – на рисунке 5.4, панели поиска комнат – на рисунке 5.5.

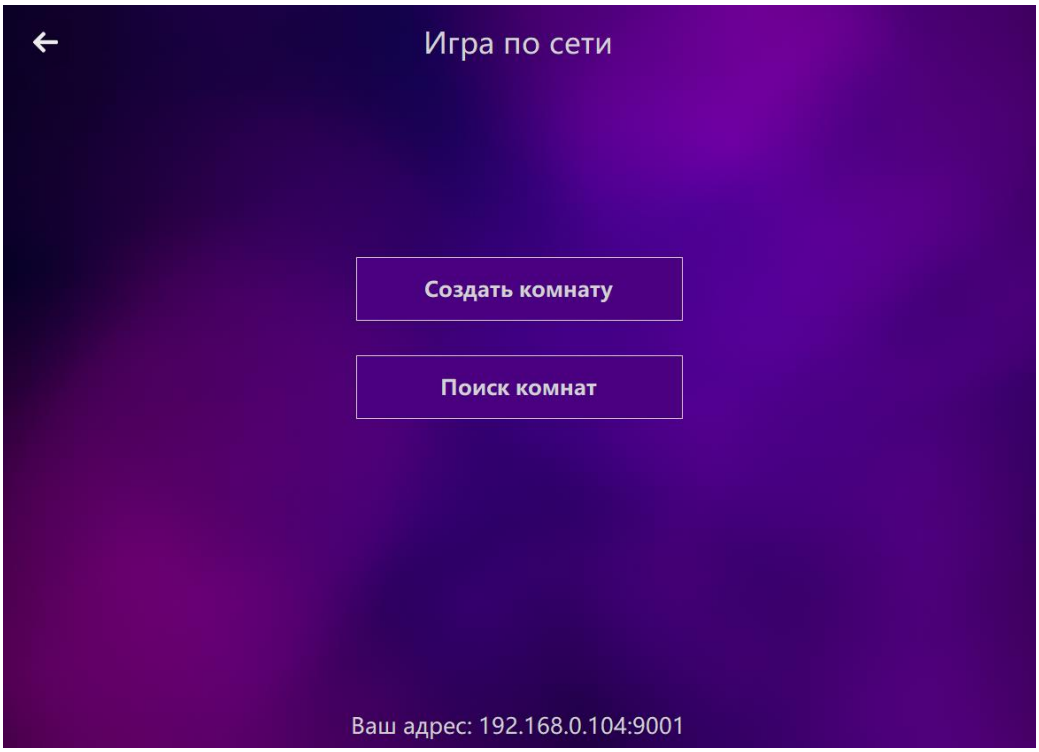

Рисунок 5.3 – Окно управления игровыми комнатами

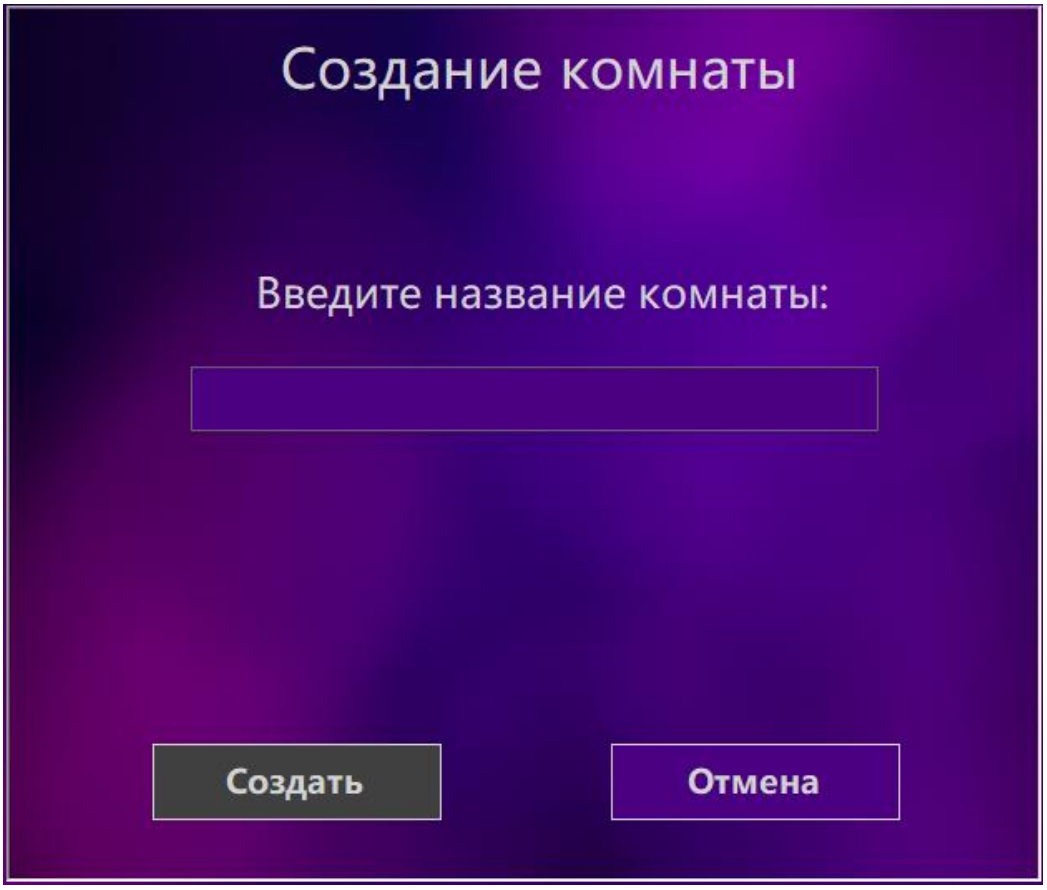

Рисунок 5.4 – Панель создания игровой комнаты

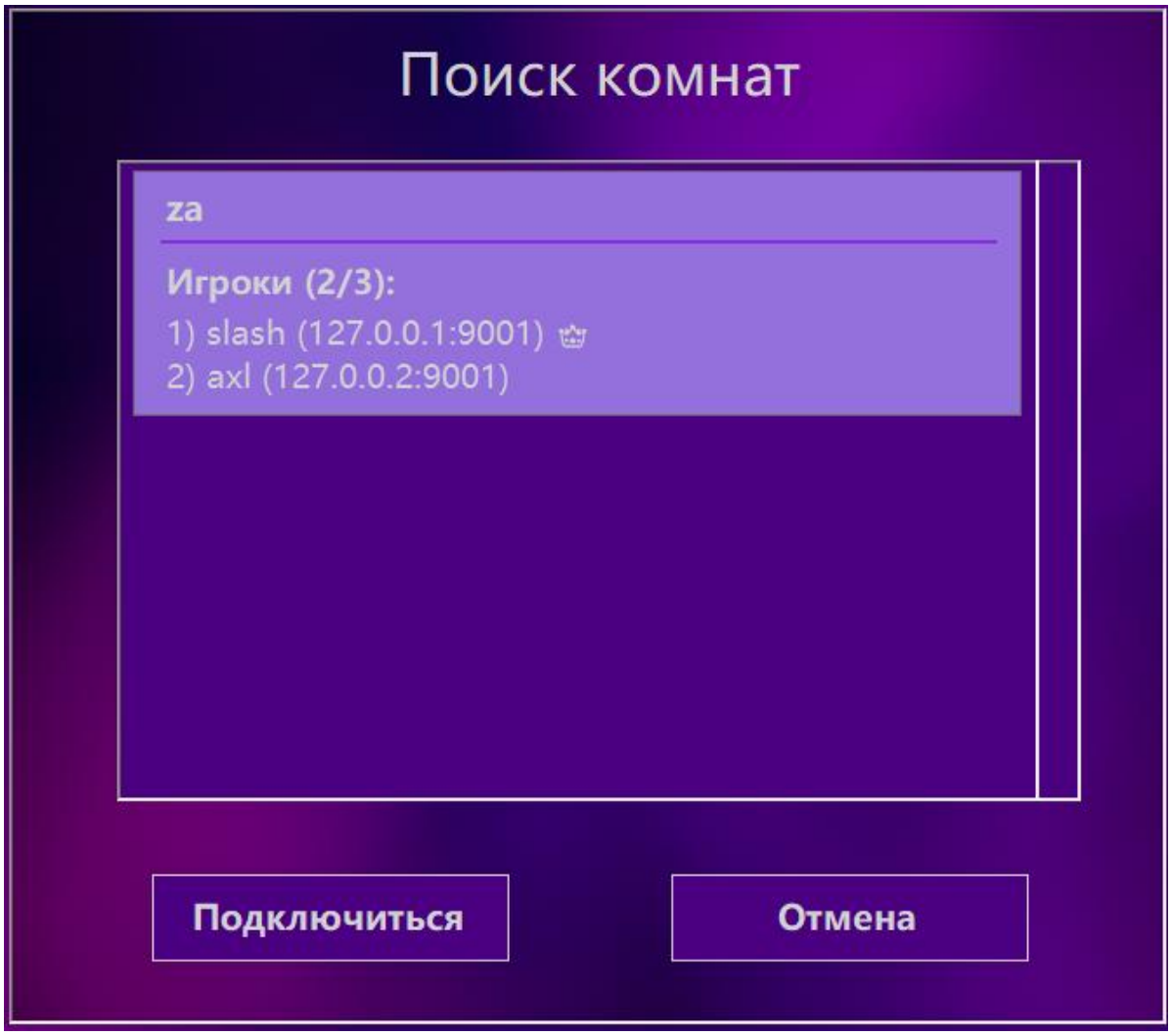

Рисунок 5.5 – Панель поиска доступных игровых комнат

5.1.3 Окно игрового процесса

Окно игрового процесса, реализованное в модуле *GameForm*, является центральным элементом приложения. Оно включает кнопки «Крутить барабан» и «Назвать слово», доступные только игроку, чей ход активен. Табло с ячейками букв загаданного слова и темой игры отображается в верхней части окна. Картинка барабана с указателем на текущий сектор, изображение ведущего и облако с его сообщениями (*lblMessageYakub*) создают атмосферу шоу. Информация об игроках – их имена, сетевые адреса и очки – отображается для всех участников. Панель с буквами (*panelLetters*) появляется, когда игроку нужно назвать букву. Внешний вид окна игрового процесса представлен на рисунке 5.6.

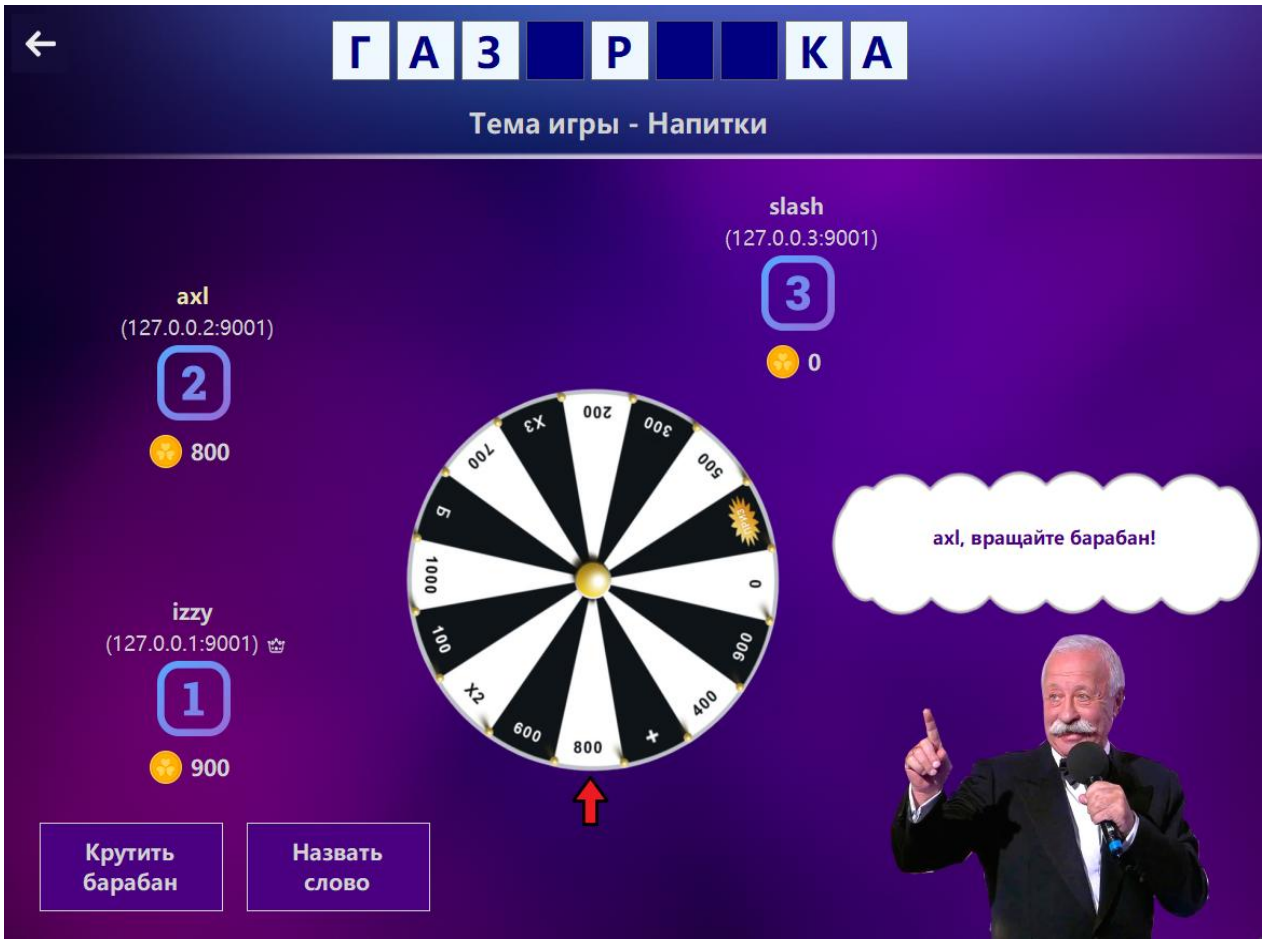

Рисунок 5.6 – Окно игрового процесса

5.1.4 Окно просмотра призов

Окно просмотра призов, реализованное в модуле *PrizesForm*, отображает список призов, доступных выбранному игроку, с их названиями, изображениями и количеством. Окно открывается из главного меню и предназначено для просмотра наград, полученных в ходе игры. Внешний вид окна просмотра призов представлен на рисунке 5.7.

Рисунок 5.7 – Окно просмотра призов

5.2 Управление программным средством

Управление программным средством осуществляется через интуитивно понятные элементы интерфейса, такие как кнопки и текстовые поля. В этом разделе описаны действия, необходимые для использования программы, включая работу с игроками, управление игровыми комнатами, участие в игре и просмотр призов.

5.2.1 Начало работы и управление игроками

После запуска приложения открывается главное окно (*MenuForm*). Для начала игры требуется выбрать или создать игрока. Нужно нажать кнопку «Сменить игрока», чтобы открыть панель управления игроками. В появившемся списке отображаются существующие игроки, сохранённые в файле *players.json*. Чтобы создать нового игрока, следует нажать кнопку «Новый», ввести имя в текстовом поле и подтвердить действие. Для выбора существующего игрока необходимо щёлкнуть по его имени в списке и нажать кнопку «ОК». Если требуется удалить игрока, нужно выделить его имя и нажать кнопку «Удалить». После выбора игрока его имя отобразится в

приветственном сообщении главного окна, и можно продолжить, нажав кнопку «Играть по сети» для перехода к окну управления комнатами.

5.2.2 Создание и подключение к игровой комнате

В окне управления игровыми комнатами (*RoomForm*) отображается сетевой адрес пользователя. Чтобы создать новую комнату, нужно нажать кнопку «Создать комнату», что откроет панель создания комнаты. Требуется ввести название комнаты в текстовом поле и нажать кнопку «Создать», чтобы стать лидером комнаты. Комната появится в локальной сети, и другие игроки смогут к ней подключиться. Если необходимо присоединиться к существующей комнате, следует нажать кнопку «Поиск комнат», чтобы открыть панель поиска. В списке доступных комнат отображаются их названия, количество игроков и данные участников. Можно выбрать комнату, щёлкнув по ней, и нажать кнопку «Подключиться». Если комната не заполнена (менее трёх игроков), пользователь присоединится, и откроется окно игрового процесса (*GameForm*). Для возврата в главное меню нужно нажать кнопку «Назад» в окне управления комнатами.

5.2.3 Участие в игровом процессе

В окне игрового процесса (*GameForm*) отображается информация о комнате, включая её название и список игроков с их именами, адресами и очками. Когда в комнате собирается три игрока, начинается обратный отсчёт, сопровождаемый сообщениями ведущего в *lblMessageYakub*, например, «До начала игры осталось – 5». После отсчёта отображается тема игры и табло с ячейками букв загаданного слова (*panelWord*). Ход передаётся первому игроку, определённому по времени подключения.

Если ход активен, нужно нажать кнопку «Крутить барабан», чтобы запустить анимацию в *pictureBoxBaraban*. После остановки барабана ведущий объявит сектор, например, «300 очков» или «ПЛЮС», через *lblMessageYakub*. Для числовых секторов (100–1000) или множителей («x2», «x3») появится панель букв (*panelLetters*). Следует щёлкнуть по букве, чтобы назвать её. Если буква есть в слове, она откроется на табло, и начислятся очки, иначе ход перейдёт к следующему игроку. Для сектора «ПЛЮС» необходимо щёлкнуть по любой закрытой ячейке на табло, чтобы открыть букву. При секторе «ПРИЗ» можно выбрать, продолжить игру или забрать приз. Для сектора «Банкрот» очки обнуляются, а для сектора «0» очки остаются без изменений, и ход передаётся дальше.

Чтобы угадать слово целиком, нужно нажать кнопку «Назвать слово», ввести ответ в появившемся поле и подтвердить. При правильном ответе игра завершится, и ведущий объявит победителя. Если ответ неверный, ход перейдёт к следующему игроку. Все действия синхронизируются через

NetworkManager, и сообщения ведущего отображаются одновременно на всех устройствах.

5.2.4 Просмотр призов

Для просмотра полученных призов следует вернуться в главное окно (*MenuForm*) и нажать кнопку «Призы». Откроется окно *PrizesForm*, где отображается список призов с их названиями, изображениями и количеством. Это окно позволяет оценить достижения для выбранного игрока. И далее, для того, чтобы вернуться в главное меню, нужно нажать кнопку «Назад».

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано программное средство «Поле Чудес», представляющее собой надёжный и удобный инструмент для организации многопользовательской сетевой викторины в локальной сети. Созданное приложение позволяет игрокам соревноваться в формате телевизионной игры, предоставляя функциональные возможности для управления игроками, создания и поиска игровых комнат, а также реализации игрового процесса с синхронизацией действий.

Разработанное приложение обладает интуитивно понятным интерфейсом, что делает его доступным для пользователей с различным уровнем технической подготовки. Реализованные функции, такие как создание игровых комнат, поиск игроков, вращение барабана и угадывание слов, обеспечивают увлекательный игровой опыт. Сетевое взаимодействие, основанное на *P2P*-архитектуре с использованием протоколов *UDP* и *TCP*, гарантирует быструю и надёжную передачу данных между участниками.

В процессе разработки были учтены современные требования к сетевым игровым приложениям, включая удобство использования и стабильность работы в локальной сети. Программа реализует структурированный подход к управлению данными игроков и вопросов, что упрощает её дальнейшее расширение и модификацию.

Таким образом, приложение «Поле Чудес» успешно решает задачу создания сетевой многопользовательской викторины, устраняя ограничения существующих аналогов, такие как отсутствие сетевого режима. Разработка данного программного средства демонстрирует эффективное применение методов анализа, проектирования и программирования для создания продукта, удовлетворяющего потребности целевой аудитории.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Поле Чудес [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D1%87%D1%83%D0%B4%D0%B5%D1%81.

[2] Поле Чудес – игра [Электронный ресурс]. – Режим доступа : https://playminigames.net/ru/game/pole-chudes.

[3] Поле Чудес Плюс [Электронный ресурс]. – Режим доступа : https://trashbox.ru/link/pole-chudes-plus-android.

[4] Поле Чудес *(HeroCraft)* [Электронный ресурс]. – Режим доступа : https://igrowiki.fandom.com/ru/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D1%87%D1%83%D0%B4%D0%B5%D1%81_(HeroCraft).

[5] Поле Чудес – отзывы [Электронный ресурс]. – Режим доступа : https://irecommend.ru/content/pole-chudes-0.

[6] *P2P* модель в играх [Электронный ресурс]. – Режим доступа : https://sky.pro/wiki/gamedev/p2p-model-v-igrah/.

[7] Преимущества и недостатки *UDP* [Электронный ресурс]. – Режим доступа : https://sky.pro/wiki/sql/preimushestva-i-nedostatki-udp/.

[8] *TCP* протокол: определение и назначение [Электронный ресурс]. – Режим доступа : https://sky.pro/wiki/sql/tcp-protokol-opredelenie-i-naznachenie/.

[9] Что такое *JSON*: отличия и преимущества [Электронный ресурс]. – Режим доступа : https://optimalgroup.ru/blog/json/.

[10] Потоки и *InvokeRequired* [Электронный ресурс]. – Режим доступа : https://www.cyberforum.ru/csharp-beginners/thread2021057.html.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода с комментариями

Файл *NetworkManager.cs*:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Net;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Поле_Чудес {
   public class NetworkManager {

      // Внутренний класс для хранения данных
      private class NetworkData {
         // Сокеты + IP
         public string LocalIp;  // Текущий IP
         public Socket UdpSocket;  // Сокет для UDP-рассылки/приёма
         public Socket TcpServerSocket;  // Сокет для приёма TCP-соединений
         // Комнаты
         public RoomInfo CurrentRoom;  // Текущая комната игрока
         public List<RoomInfo> AvailableRooms;  // Список доступных комнат
         // Флаги
         public bool IsLeader;  // Является ли игрок лидером
         public bool IsRunning;  // Работает ли NetworkManager
         // Порты
         public int UdpPort;  // Текущий UDP-порт (по умолчанию 9000)
         public int TcpPort;  // Текущий TCP-порт (по умолчанию 9001)
         // Потоки
         public Thread UdpReceiveThread;  // Поток для приёма UDP-пакетов
         public Thread TcpAcceptThread;  // Поток для принятия TCP-соединений
         public Thread UdpBroadcastThread;  // Поток для рассылки UDP (только
для лидера)
      }

      // Статическое поле для данных (сокеты, списки, порты)
      private static readonly NetworkData Manager = new NetworkData {
```

```
        CurrentRoom = null,
        AvailableRooms = new List<RoomInfo>(),
        IsRunning = false,  // флаг - активна ли сеть
        IsLeader = false,
        LocalIp = null
    };

    // Информция о комнате
    public class RoomInfo {
        public string Name { get; set; }
        public string Ip { get; set; }
        public int Port { get; set; }  // TCP-порт лидера
        public int PlayerCount { get; set; }
        public string Status { get; set; }
        public List<PlayerInfo> Players { get; set; } = new List<PlayerInfo>(); //
Список игроков комнаты
        public Question Question { get; set; }  // Загаданный вопрос
    }

    // Информация об игроке
    public class PlayerInfo {
        public string Name { get; set; }
        public string Ip { get; set; }
        public int Port { get; set; }  // TCP-порт игрока
        public Socket Socket { get; set; }
        public DateTime JoinTime { get; set; }
    }

    // Публичный доступ к нашему адресу для RoomForm
    public static string LocalAddress =>
$"{Manager.LocalIp}:{Manager.TcpPort}";

    // Событие для уведомления об обновлении списка комнат
    public static event Action RoomsUpdated;

    // Событие для уведомления UI о новом игроке
    public static event Action<PlayerInfo> PlayerJoined;

    // Событие для уведомления UI о получении сектора барабана
    public static event Action<string> SpinBarabanReceived;
```

```
    // Событие для уведомления UI о получении названной буквы
    public static event Action<string> SelectedLetterReceived;

    // Публичный метод для получения копии списка комнат
    public static List<RoomInfo> GetAvailableRooms() {
        lock (Manager.AvailableRooms) {
            // !!! ДОБАВИТЬ ПОТОМ ФИЛЬТРЫ -- Where(r => r.PlayerCount < 3
&& r.Status == "Waiting")
            return Manager.AvailableRooms.ToList(); // Копия для безопасности
        }
    }

    // Публичный метод для получения текущей комнаты
    public static RoomInfo GetCurrentRoom() {
        return Manager.CurrentRoom;
    }

    // Запуск Manager
    public static void Start() {
        // Устанавливаем флаг, что сеть активна
        Manager.IsRunning = true;

        // Проверяем порты
        CheckUdpPort(9000);
        CheckTcpPort(9001);

        // Получаем и преобразуем IP
        Manager.LocalIp = GetLocalIp();
        IPAddress ipAddress;
        try {
            ipAddress = IPAddress.Parse(Manager.LocalIp);
        }
        catch (FormatException ex) {
            throw new Exception($"Неверный формат IP {Manager.LocalIp}:
{ex.Message}");
        }

        // Привязываем UDP-сокет
        Manager.UdpPort = 9000;
        try {
```

```
            Manager.UdpSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
            Manager.UdpSocket.EnableBroadcast = true;
            Manager.UdpSocket.Bind(new IPEndPoint(ipAddress,
Manager.UdpPort));
        }
        catch (SocketException ex) {
            Manager.UdpSocket?.Close();
            throw new Exception($"IP {Manager.LocalIp} уже занят другим
процессом: {ex.Message}");
        }

        // Привязываем TCP-сокет
        Manager.TcpPort = 9001;
        try {
            Manager.TcpServerSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            Manager.TcpServerSocket.Bind(new IPEndPoint(ipAddress,
Manager.TcpPort));
            Manager.TcpServerSocket.Listen(10);
        }
        catch (SocketException ex) {
            Manager.UdpSocket?.Close();
            Manager.TcpServerSocket?.Close();
            throw new Exception($"IP {Manager.LocalIp} уже занят другим
процессом: {ex.Message}");
        }

        // Запускаем поток для принятия UDP-пакетов (обнаружение комнат)
        Manager.UdpReceiveThread = new Thread(ReceiveUdpBroadcasts);
        Manager.UdpReceiveThread.Start();
    }

    // Проверка доступности UDP-порта
    private static void CheckUdpPort(int port) {
        Socket socket = null;  // создаем временный тестовый сокет
        try {
            socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp);
            socket.Bind(new IPEndPoint(IPAddress.Any, port));
            socket.Close(); // Порт свободен, явно закрываем
```

```
        }
        catch (SocketException ex) {
            socket?.Close();
            throw new Exception($"Порт {port} для UDP занят: {ex.Message}");
        }
    }

    // Проверка доступности TCP-порта
    private static void CheckTcpPort(int port) {
        Socket socket = null;  // создаем временный тестовый сокет
        try {
            socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            socket.Bind(new IPEndPoint(IPAddress.Any, port));
            socket.Listen(10);
            socket.Close(); // Порт свободен, явно закрываем
        }
        catch (SocketException ex) {
            socket?.Close();
            throw new Exception($"Порт {port} для TCP занят: {ex.Message}");
        }
    }

    // Получение локального IP
    public static string GetLocalIp() {
        // Получаем аргументы командной строки
        string[] args = Environment.GetCommandLineArgs();
        string ipArg = null;

        // Ищем параметр "--ip=127.0.0.1"
        foreach (var arg in args) {
            if (arg.StartsWith("--ip="))
                ipArg = arg.Substring(5);
        }
        // Возвращаем ip из параметра CMD - если он есть
        if (!string.IsNullOrEmpty(ipArg)) return ipArg;

        // Если аргумента нет, ищем IP в локальной сети
        var host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList) {
            if (ip.AddressFamily == AddressFamily.InterNetwork)
```

```
            return ip.ToString();
        }
        // Запасной вариант
        return "127.0.0.1";
    }

    // Создание комнаты (+ затирает старую комнату)
    public static void CreateRoom(string roomName) {



        // Устанавливаем игрока как лидера
        Manager.IsLeader = true;

        // Генерируем случайный вопрос
        var question = QuestionControl.GetRandomQuestion();

        // Инициализируем CurrentRoom
        Manager.CurrentRoom = new RoomInfo {
            Name = roomName,
            Ip = Manager.LocalIp,
            Port = Manager.TcpPort,
            PlayerCount = 1,
            Status = "Waiting",
            Players = new List<PlayerInfo> {
                // Добавляем текущего игрока в список Players
                new PlayerInfo {
                    Name = MenuForm.ActivePlayer.Name,
                    Ip = Manager.LocalIp,
                    Port = Manager.TcpPort,
                    Socket = null,
                    JoinTime = DateTime.Now
                }
            },
            Question = question
        };

        // Завершаем предыдущий поток рассылки, если он существует
        if (Manager.UdpBroadcastThread != null &&
Manager.UdpBroadcastThread.IsAlive) {
            try {
```

```
                Manager.UdpBroadcastThread.Interrupt();
                Manager.UdpBroadcastThread.Join(100); // Ждём завершения до 100
мс
            }
            catch { }
        }

        // Завершаем предыдущий поток принятия TCP, если он существует
        if (Manager.TcpAcceptThread != null &&
Manager.TcpAcceptThread.IsAlive) {
            try {
                Manager.TcpAcceptThread.Interrupt();
                Manager.TcpAcceptThread.Join(100);
            }
            catch { }
        }

        // Запускаем поток для принятия TCP-соединений
        Manager.TcpAcceptThread = new Thread(AcceptTcpClients);
        Manager.TcpAcceptThread.Start();

        // Запускаем поток для отправки UDP-рассылки (только для лидера)
        Manager.UdpBroadcastThread = new Thread(SendUdpBroadcast);
        Manager.UdpBroadcastThread.Start();
    }


    // Отправка UDP-рассылки
    private static void SendUdpBroadcast() {

        while (Manager.IsRunning && Manager.IsLeader) {
            try {
                // Формируем строку с данными игроков: Name|Ip|Port|JoinTime;...
                var playersData = string.Join(";",
Manager.CurrentRoom.Players.Select(p =>
                    $"{p.Name}|{p.Ip}|{p.Port}|{p.JoinTime.Ticks}"));

                // Формируем строку с вопросом: Topic|Word
                var questionData = Manager.CurrentRoom.Question != null
```

```
                ?
$"{Manager.CurrentRoom.Question.Topic}|{Manager.CurrentRoom.Question.Word}": "";

            // Отправляем UDP-пакет на 255.255.255.255:9000 с информацией о комнате
            string message = $"Room:{Manager.CurrentRoom.Name},IP:{Manager.LocalIp},Port:{Manager.TcpPort}," +

$"Players:{Manager.CurrentRoom.Players.Count}/3,Status:Waiting,PlayerData:{playersData},Question:{questionData}";
            byte[] data = Encoding.UTF8.GetBytes(message);
            Manager.UdpSocket.SendTo(data, new IPEndPoint(IPAddress.Broadcast, 9000));  // Рассылаем на стандартный порт 9000
            Thread.Sleep(2000);  // Рассылка каждые 2 секунды
        }
        catch { }
    }
}

// Приём UDP-пакетов
private static void ReceiveUdpBroadcasts() {
    byte[] buffer = new byte[4096];  // 1024 раньше
    EndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, 0);

    while (Manager.IsRunning) {
        try {
            // Принимаем UDP-пакеты
            int bytesRead = Manager.UdpSocket.ReceiveFrom(buffer, ref remoteEndPoint);
            string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);

            // Парсим пакеты с информацией о комнатах
            if (message.StartsWith("Room:")) {
                var parts = message.Split(',');
                if (parts.Length >= 7 && parts[0].StartsWith("Room:") && parts[1].StartsWith("IP:") &&
                    parts[2].StartsWith("Port:") && parts[3].StartsWith("Players:") && parts[4].StartsWith("Status:") &&
```

```
                parts[5].StartsWith("PlayerData:") &&
parts[6].StartsWith("Question:")) {

                string name = parts[0].Substring(5);
                string ip = parts[1].Substring(3);
                if (!int.TryParse(parts[2].Substring(5), out int port)) continue;
                string playersPart = parts[3].Substring(8);
                if (!int.TryParse(playersPart.Split('/')[0], out int playerCount))
continue;
                string status = parts[4].Substring(7);
                string playerData = parts[5].Substring(11);
                string questionData = parts[6].Substring(9);

                var players = new List<PlayerInfo>();
                if (!string.IsNullOrEmpty(playerData)) {
                    var playerEntries = playerData.Split(';',
(char)StringSplitOptions.RemoveEmptyEntries);
                    foreach (var entry in playerEntries) {
                        var fields = entry.Split('|');
                        if (fields.Length == 4 &&
                            !string.IsNullOrEmpty(fields[0]) &&
                            !string.IsNullOrEmpty(fields[1]) &&
                            int.TryParse(fields[2], out int playerPort) &&
                            long.TryParse(fields[3], out long ticks)) {
                            players.Add(new PlayerInfo {
                                Name = fields[0],
                                Ip = fields[1],
                                Port = playerPort,
                                JoinTime = new DateTime(ticks),
                                Socket = null
                            });
                        }
                    }
                }

                // Парсим вопрос
                Question question = null;
                if (!string.IsNullOrEmpty(questionData)) {
                    var questionParts = questionData.Split('|');
                    if (questionParts.Length == 2) {
                        question = new Question {
```

```
                        Topic = questionParts[0],
                        Word = questionParts[1]
                    };
                }
            }

            // Обновляет или добавляет комнату в список доступных комнат AvailableRooms
            lock (Manager.AvailableRooms) {
                var room = Manager.AvailableRooms.FirstOrDefault(r => r.Name == name);
                if (room == null) {
                    Manager.AvailableRooms.Add(new RoomInfo {
                        Name = name,
                        Ip = ip,
                        Port = port,
                        PlayerCount = playerCount,
                        Status = status,
                        Players = players,
                        Question = question
                    });
                }
                else {
                    // Обновляем IP/Port при смене лидера
                    room.Ip = ip;
                    room.Port = port;
                    room.PlayerCount = playerCount;
                    room.Status = status;
                    room.Players = players;
                    room.Question = question;
                }
            }

            RoomsUpdated?.Invoke();  // Уведомляем об обновлении списка комнат
        }
    }
    catch { }
  }
}
```

```
        // Подключает игрока к существующей комнате
        public static void JoinRoom(RoomInfo room) {
            // Проверяем валидность комнаты
            if (room == null || room.Players.Count == 0)
                throw new Exception("Комната не выбрана или пуста");

            // Проверяем, не заполнена ли комната
            if (room.PlayerCount >= 3)
                throw new Exception("Комната заполнена");

            // Инициализируем текущую комнату на основе выбранной
            Manager.CurrentRoom = new RoomInfo {
                Name = room.Name,
                Ip = room.Ip,
                Port = room.Port,
                PlayerCount = room.PlayerCount + 1,
                Status = room.Status,
                Players = new List<PlayerInfo>(room.Players),
                Question = room.Question
            };

            // Добавляем себя в список игроков
            var self = new PlayerInfo {
                Name = MenuForm.ActivePlayer.Name,
                Ip = Manager.LocalIp,
                Port = Manager.TcpPort,
                Socket = null,
                JoinTime = DateTime.Now
            };
            Manager.CurrentRoom.Players.Add(self);

            // Подключаемся к каждому игроку в комнате, кроме самих себя
            foreach (var player in Manager.CurrentRoom.Players.Where(p => p.Ip !=
Manager.LocalIp)) {
                try {
                    // Создаём TCP-сокет для подключения и инициируем подключение
к нему
                    Socket socket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                    socket.Connect(player.Ip, player.Port);
```

```
                // Сохраняем сокет для игрока из текущей комнаты
                lock (Manager.CurrentRoom) { player.Socket = socket; }

                // Отправляем сообщение Join с информацией о себе
                SendMessage(socket, 1, $"{self.Name}:{self.Ip}:{self.Port}");

                // Запускаем поток для приёма сообщений от этого игрока
                Thread receiveThread = new Thread(() => ReceiveMessages(socket));
                receiveThread.Start();
            }
            catch {

            }
        }

        // Запускаем поток для принятия TCP-соединений
        Manager.TcpAcceptThread = new Thread(AcceptTcpClients);
        Manager.TcpAcceptThread.Start();

    }

    // Отправляет результат вращения барабана другим игрокам
    public static void SendSpinResult(string sector) {
        lock (Manager.CurrentRoom) {
            // Проверка, не пустая ли комната
            if (Manager.CurrentRoom == null || Manager.CurrentRoom.Players ==
null) return;

            // Отправляем сообщение всем игрокам в комнате, кроме самих себя
            foreach (var player in Manager.CurrentRoom.Players.Where(p => p.Ip
!= Manager.LocalIp)) {
                if (player.Socket != null)
                    SendMessage(player.Socket, 2, sector); // Тип 2: SpinBaraban
            }
        }
    }

    // Отправляет названную букву другим игрокам
    public static void SendSelectedLetter(string letter) {
        lock (Manager.CurrentRoom) {
```

```
            // Проверка, не пустая ли комната
            if (Manager.CurrentRoom == null || Manager.CurrentRoom.Players == null) return;

            // Отправляем сообщение всем игрокам в комнате, кроме самих себя
            foreach (var player in Manager.CurrentRoom.Players.Where(p => p.Ip != Manager.LocalIp)) {
                if (player.Socket != null)
                    SendMessage(player.Socket, 3, letter); // Тип 3: SelectedLetter
            }
        }
    }

    // Принятие TCP-соединений от других игроков
    private static void AcceptTcpClients() {
        while (Manager.IsRunning) {
            try {
                // Принимаем входящие TCP-соединения
                Socket clientSocket = Manager.TcpServerSocket.Accept();

                // Запускаем поток для обработки сообщений от клиента
                Thread receiveThread = new Thread(() => ReceiveMessages(clientSocket));
                receiveThread.Start();
                // Каждый поток независим и привязан к своему сокету, старые потоки не закрываются
            }
            catch {
                // Игнорируем ошибки принятия
            }
        }
    }

    // Обрабатывает входящие сообщения от других игроков
    private static void ReceiveMessages(Socket socket) {
        // Буфер для заголовка сообщения (1 байт типа + 4 байта длины)
        byte[] headerBuffer = new byte[5];
        try {
            // Продолжаем читать сообщения, пока сеть активна
            while (Manager.IsRunning) {
                // Читаем заголовок и помещаем в буфер headerBuffer
```

```
            int bytesRead = socket.Receive(headerBuffer);
            if (bytesRead < 5) break;

            // Извлекаем тип сообщения и длину данных
            byte type = headerBuffer[0];
            int length = IPAddress.NetworkToHostOrder(BitConverter.ToInt32(headerBuffer, 1));

            // Читаем данные сообщения и помещаем в буфер dataBuffer
            byte[] dataBuffer = new byte[length];
            int totalRead = 0;
            while (totalRead < length) {
               bytesRead = socket.Receive(dataBuffer, totalRead, length - totalRead, SocketFlags.None);
               if (bytesRead == 0) throw new SocketException();
               totalRead += bytesRead;
            }

            // Преобразуем данные в строку
            string message = Encoding.UTF8.GetString(dataBuffer);

            // Обрабатываем сообщение по типу
            switch (type) {
               case 1: // Join: новый игрок подключается
                  if (message.Contains(':')) {
                     // Парсим сообщение
                     var parts = message.Split(':');
                     if (parts.Length == 3 && !string.IsNullOrEmpty(parts[0]) &&
                        !string.IsNullOrEmpty(parts[1]) && int.TryParse(parts[2], out int port)) {

                        // Синхронизируем доступ к CurrentRoom
                        lock (Manager.CurrentRoom) {
                           if (Manager.CurrentRoom != null && Manager.CurrentRoom.PlayerCount < 3) {
                              // Создаём объект нового игрока
                              var newPlayer = new PlayerInfo {
                                 Name = parts[0],
                                 Ip = parts[1],
                                 Port = port,
                                 Socket = socket,
```

```
                        JoinTime = DateTime.Now
                    };

                    // Проверяем, нет ли уже этого игрока
                    if (!Manager.CurrentRoom.Players.Any(p => p.Name
== newPlayer.Name &&
                        p.Ip == newPlayer.Ip && p.Port ==
newPlayer.Port)) {
                        // Добавляем игрока в список
                        Manager.CurrentRoom.Players.Add(newPlayer);
                        Manager.CurrentRoom.PlayerCount =
Manager.CurrentRoom.Players.Count;

                        // Уведомляем локальный UI о новом игроке
                        PlayerJoined?.Invoke(newPlayer);
                    }
                }
            }
        }
        break;
    case 2: // SpinBaraban: получен сектор барабана
        SpinBarabanReceived?.Invoke(message); // Уведомляем
локальный UI о вращении барабана до сектора
        break;
    case 3: // SelectedLetter: получена названная буква
        SelectedLetterReceived?.Invoke(message); // Уведомляем
локальный UI о назывании буквы
        break;
    }
  }
  catch {

    // Закрываем сокет при ошибке или отключении
    socket.Close();
  }
}

// Отправляет байтовое сообщение через сокет
private static void SendMessage(Socket socket, byte type, string message) {
```

```
            // Создаём сообщение
            byte[] data = CreateMessage(type, message);
            try {
                // Отправляем сообщение
                socket.Send(data);
            }
            catch {

            }
        }

        // Создаёт байтовое сообщение с заголовком [type:1][length:4][data]
        private static byte[] CreateMessage(byte type, string message) {
            // Преобразуем сообщение в байты
            byte[] data = Encoding.UTF8.GetBytes(message);

            // Создаём заголовок
            byte[] header = new byte[5];
            header[0] = type;

Array.Copy(BitConverter.GetBytes(IPAddress.HostToNetworkOrder(data.Length)),
0, header, 1, 4);

            // Объединяем заголовок и данные
            byte[] result = new byte[5 + data.Length];
            Array.Copy(header, 0, result, 0, 5);
            Array.Copy(data, 0, result, 5, data.Length);
            return result;
        }

        // Остановка Manager
        public static void Stop() {
            // Отключаем флаг активности сети (останавливает циклы в
потоках)
            Manager.IsRunning = false;

            // Закрываем все сокеты игроков
            if (Manager.CurrentRoom != null) {
                lock (Manager.CurrentRoom) {
                    foreach (var player in Manager.CurrentRoom.Players)
                        player.Socket?.Close();
```

```
            }
            // Очищаем текущую комнату после закрытия сокетов
            Manager.CurrentRoom = null;
        }

        // Останавливаем UDP-рассылку и закрываем UDP-сокет
        Manager.UdpBroadcastThread?.Interrupt();
        Manager.UdpSocket?.Close();

        // Останавливаем TCP-сервер и закрываем TCP-сокет
        Manager.TcpAcceptThread?.Interrupt();
        Manager.TcpServerSocket?.Close();

        // Останавливаем приём UDP
        Manager.UdpReceiveThread?.Interrupt();

        // Очищаем список AvailableRooms
        lock (Manager.AvailableRooms) {
            Manager.AvailableRooms.Clear();
        }
    }
}
}
```