Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**РАЗРАБОТКА СЕТЕВОЙ КОМПЮТЕРНОЙ ИГРЫ «PING PONG ONLINE»**

БГУИР КП 6-05-0612-01-007 ПЗ

Студент Веренич В.Г.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[Введение 4](#_Toc199268148)

[1 Анализ предметной области 5](#_Toc199268149)

[1.1 Обзор аналогов 5](#_Toc199268150)

[1.2 Постановка задачи 6](#_Toc199268151)

[2 Проектирование программного средства 7](#_Toc199268152)

[2.1 Структура программы 7](#_Toc199268153)

[2.2 Проектирование интерфейса программного средства 9](#_Toc199268154)

[2.3 Проектирование функционала программного средства 11](#_Toc199268155)

[3 Разработка программного средства 15](#_Toc199268156)

[3.1 Движение мяча и управление игровым процессом 15](#_Toc199268157)

[3.2 Управление игровыми сценами и сессией 17](#_Toc199268158)

[3.3 Работа с игровыми событиями 18](#_Toc199268159)

[3.4 Работа с игровыми данными 20](#_Toc199268160)

[4 Тестирование программного средства 24](#_Toc199268161)

[5 Руководство пользователя 26](#_Toc199268162)

[5.1 Интерфейс программного средства 26](#_Toc199268163)

[5.2 Управление программным средством 28](#_Toc199268164)

[Заключение 30](#_Toc199268165)

[Список Использованных Источников 31](#_Toc199268166)

[Приложение А. Исходный Код Программы 32](#_Toc199268167)

# ВВЕДЕНИЕ

В современном мире информационных технологий и цифровых развлечений компьютерные игры играют значимую роль не только как средство досуга, но и как важный инструмент обучения, социализации и развития логического мышления. Особенно популярными в последние годы стали многопользовательские онлайн-игры, обеспечивающие взаимодействие между участниками в реальном времени. Такие игры требуют не только качественной графики и продуманного геймплея, но и эффективной сетевой архитектуры, способной обеспечивать стабильную синхронизацию данных между клиентами.

Одним из классических и в то же время увлекательных примеров аркадных игр является Ping Pong — настольный теннис, реализуемый в виде простой двумерной игры. Несмотря на кажущуюся простоту, данная игра является отличной платформой для изучения основ игровой логики, управления объектами, работы с пользовательским интерфейсом, а также основ сетевого взаимодействия между пользователями.

Целью данного курсового проекта является создание многопользовательской игры «Ping Pong Online» с использованием игрового движка Unity. Основное внимание в ходе разработки уделено сетевому взаимодействию между двумя игровыми клиентами через внешний TCP-сервер. Один из игроков управляет левой ракеткой, второй — правой. Позиции мяча, ракеток, а также текущий счёт и игровые события синхронизируются через сервер, что обеспечивает корректную совместную игру.

В процессе реализации проекта предполагается решить ряд прикладных задач: создать удобный интерфейс, реализовать физику движения мяча и его отскоков, организовать управление ракетками с обеих сторон, обеспечить бесперебойную передачу данных между клиентами, а также предусмотреть отображение счёта и завершение игры при достижении победных очков.

Проект «Ping Pong Online» позволяет не только освоить инструменты Unity и работу с TCP-протоколами, но и на практике применить принципы объектно-ориентированного программирования, проектирования архитектуры многопользовательских приложений, управления потоками и обработки сетевых сообщений. Таким образом, разработка данной игры служит важным шагом в профессиональном становлении программиста, ориентированного на сферу геймдева и сетевого взаимодействия.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Игры жанра «настольный теннис» (Ping Pong или Pong) являются одними из первых в истории компьютерных развлечений. Классическая версия Pong была разработана компанией Atari в 1972 году и стала настоящим прорывом в области интерактивных электронных игр. С тех пор концепция игры была многократно адаптирована и переработана, как в виде однопользовательских приложений, так и в виде многопользовательских онлайн-игр.

В современном мире существует множество аналогичных реализаций игры пинг-понг:

– Pong Online — простейшие HTML5- или Flash-версии, запускаемые в браузере, позволяющие играть вдвоём через интернет или локальную сеть. Как правило, они имеют минималистичный интерфейс и ограниченный функционал.

– Table Tennis Touch, Ping Pong Fury — мобильные аркадные симуляторы настольного тенниса с элементами реалистичной физики, рейтинговой системой и разнообразными режимами игры, включая сетевые.

– Unity-игры из Asset Store — существуют готовые шаблоны игр Pong с базовой физикой и управлением. Однако в большинстве таких решений отсутствует полноценная поддержка клиент-серверной синхронизации, и они ориентированы на локальную игру или простую реализацию с использованием высокоуровневых сетевых API (например, Unity Netcode).

Тем не менее, большая часть подобных решений имеет ряд ограничений:

– привязка к конкретной платформе или движку;

– отсутствие гибкой архитектуры для расширения;

– ограниченная поддержка сетевого взаимодействия через TCP (в основном используются UDP или WebSocket, либо встроенные решения, недоступные для детальной настройки);

– невозможность использовать собственный сервер или управлять логикой синхронизации.

В связи с этим, возникает необходимость в создании собственной реализации игры Ping Pong Online, которая не только повторяет классическую механику, но и предоставляет гибкую сетевую архитектуру с внешним TCP-сервером, позволяющим обмениваться данными между игроками без использования сторонних библиотек или API. Это особенно актуально в образовательной среде, где важно понимать и контролировать каждый этап взаимодействия между клиентом и сервером.

## Постановка задачи

Основной целью курсового проекта является разработка многопользовательской аркадной игры «Ping Pong Online» с использованием Unity и внешнего TCP-сервера. Игра должна позволять двум игрокам подключаться к серверу и синхронно участвовать в игровом процессе: один управляет левой ракеткой, другой — правой. Мяч должен двигаться по экрану с учётом столкновений, а текущий счёт должен обновляться и отображаться на обоих клиентах.

Для достижения этой цели необходимо решить следующие задачи:

– Разработать архитектуру клиент-серверного взаимодействия. Создать отдельный сервер на языке C# (или другом подходящем), принимающий соединения от клиентов и обеспечивающий двустороннюю синхронизацию игровых данных: координат мяча, ракеток, счёта и игровых событий.

– Создать два независимых клиента на Unity. Клиенты должны иметь одинаковую игровую сцену, но различаться по логике управления (игрок 1 управляет левой ракеткой, игрок 2 — правой). Управление должно быть строго разделено.

– Реализовать механизм синхронизации игрового состояния. При помощи TCP-протокола сервер должен регулярно рассылать актуальные координаты мяча и ракеток, а также данные о счёте, всем подключённым клиентам.

– Обеспечить корректную обработку игровых событий: столкновения мяча с ракетками и границами поля, начисление очков, переход к экрану завершения игры при достижении 11 баллов одним из игроков.

– Сделать возможным перезапуск игры без потери соединения. После завершения раунда игроки должны иметь возможность начать новую игру с тем же подключением к серверу.

– Разработать пользовательский интерфейс. Интерфейс должен включать информационные элементы (счёт, статус игры) и кнопки управления (перезапуск, выход и пр.).

Дополнительно рекомендуется предусмотреть масштабируемость решения: возможность подключения большего числа игроков в режиме наблюдателя, реализацию простой системы матчмейкинга, а также возможность публикации игры для различных платформ (ПК, WebGL, Android).

В рамках курсового проекта основное внимание уделяется низкоуровневому сетевому взаимодействию, точной синхронизации игровых состояний и стабильности соединений — что делает проект не только интересным, но и крайне полезным для практического освоения клиент-серверной архитектуры в геймдеве.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано 18 модулей, каждый из которых выполняет определённую функцию в рамках архитектуры клиента. Эти модули обеспечивают корректную работу логики игры, сетевого взаимодействия, управления игроками, обработку физики и пользовательского интерфейса:

– Server – модуль, реализующий отдельный TCP-сервер на стороне сервера. Обеспечивает приём соединений от клиентов, передачу игровых данных (положение мяча, ракеток, счёт) и синхронизацию состояния между подключёнными участниками;

– BallBounce – модуль, отвечающий за обработку столкновений мяча с ракетками, границами игрового поля и другими объектами сцены. Реализует логику отскока мяча на основе направления и физики;

– BallManager – модуль, отвечающий за создание и начальное размещение мяча в основной игровой сцене. Может быть использован для повторной инициализации мяча после каждого розыгрыша;

– BallMovement – модуль, управляющий движением мяча. Отвечает за обновление его позиции, расчёт направления и скорости, а также взаимодействие с другими модулями при изменении направления движения;

– ChangeScene – модуль, отвечающий за переход между сценами (например, из меню в игру, из игры на экран окончания). Используется также для загрузки сцены повторного матча без отключения от сервера;

– DestroySoundEffect– модуль, воспроизводящий звуковой эффект при столкновении мяча с краями поля или ракеткой. Также может быть задействован при завершении раунда или начале нового;

– GameManager – основной управляющий модуль игрового процесса. Контролирует текущий счёт, победные условия, завершение игры, а также взаимодействует с UI;

– GameManagerInitializer – модуль, отвечающий за инициализацию параметров игры (например, установка начального счёта, сброс состояний при перезапуске и т.д.);

– NetworkClient– модуль, реализующий подключение клиента к серверу. Обеспечивает отправку и приём сетевых сообщений, таких как позиция игрока, мяча и текущий счёт;

– NetworkManager – вспомогательный модуль, координирующий работу сетевых компонентов клиента. Отвечает за организацию соединения, обработку сообщений, управление потоками;

– NetworkPlayerSync – модуль, реализующий синхронизацию позиций ракеток между клиентами. Отправляет координаты локального игрока на сервер и обновляет отображение позиции другого игрока по полученным данным;

– PauseMenu – модуль управления игровым меню паузы. Предоставляет возможность приостановить игру, перезапустить матч, вернуться в главное меню или выйти из приложения;

– Player1 – модуль, реализующий управление первой ракеткой (левый игрок). Обрабатывает ввод с клавиатуры или другого контроллера, отправляет данные на сервер;

– Player2 – модуль, реализующий управление второй ракеткой (правый игрок). Аналогично Player1, обрабатывает ввод и передаёт координаты в сеть;

– ScoreManager – модуль, управляющий счётом игры. Отслеживает количество очков у каждого игрока, обновляет UI, проверяет условия победы;

– SoundManager – модуль, централизованно управляющий всеми звуковыми эффектами в игре: от звука столкновений до фоновой музыки и системных звуков интерфейса;

– UnityMainThreadDispatcher – модуль, позволяющий выполнять вызовы из сетевых или фоновых потоков в основном Unity-потоке. Это необходимо для безопасного обновления UI и объектов сцены из кода, работающего в других потоках.

## 2.2 Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят стандартный внешний вид игры «Ping Pong».

2.2.1 Главное меню

Главное меню приложения содержит на экране функционал для подключения к серверу и непосредственно запуску основной игровой сцены. В верхней части окна будет находится большая надпись с названием игры, ниже будут находиться кнопки: «Играть как хост», «Играть как клиент», «Настройки», «Выход». Макет главного меню представлен на рисунке 2.1.

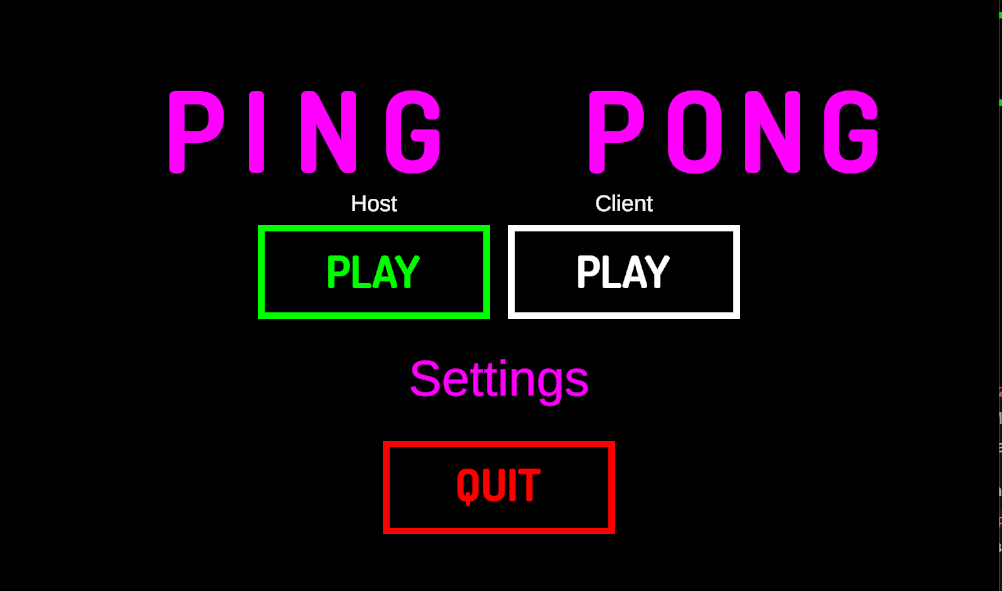


Рисунок 2.1 – Главное меню приложения

2.2.2 Основное окно игры

В основном окне игры будут представлены 2 ракетки для игроков, мячик для игры, в верхней части окна будет кнопка «Pause», которая отвечает за остановку игры, счёт игрового процесса, а также будут предствлены верхняя и нижняя границы окна, чтобы шарик от них отскакивал.

Внешний вид основного окна игры представлен на рисунке 2.2.

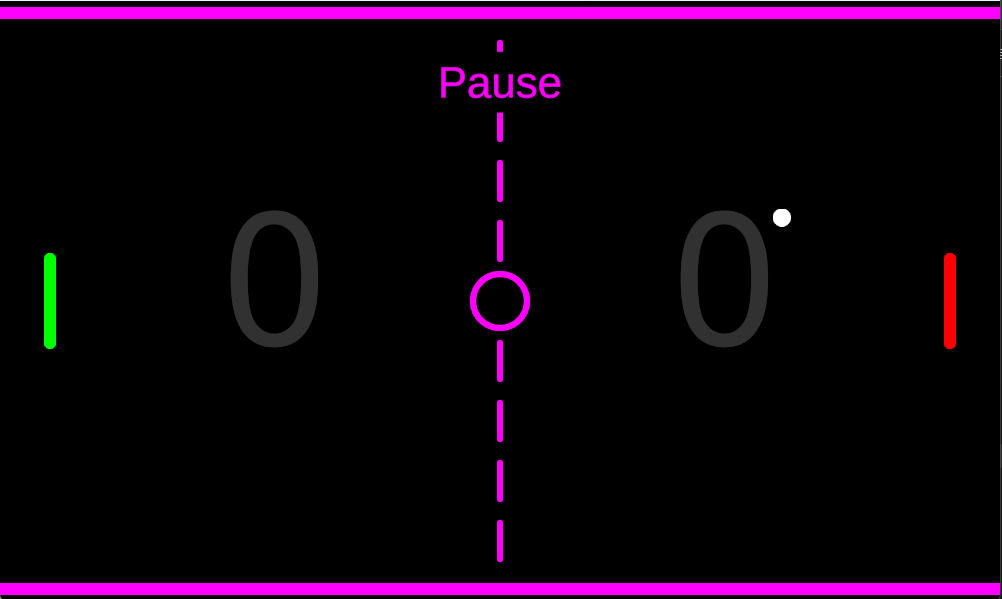


Рисунок 2.2 – Элементы управления веб-браузером

2.2.3 Окно настроек игры

Окно настроек будет состоять из двух частей. В верхней части окна будет находится большая надпись «Settings». Центральная часть окна будет представляет «Slider», который будет отвечать за регулировку громкости музыки в игре. В нижней части окна будет представлена кнопка «Main Menu», которая отвечает за возвращение в главное меню. Макет окна настроек представлен на рисунке 2.3

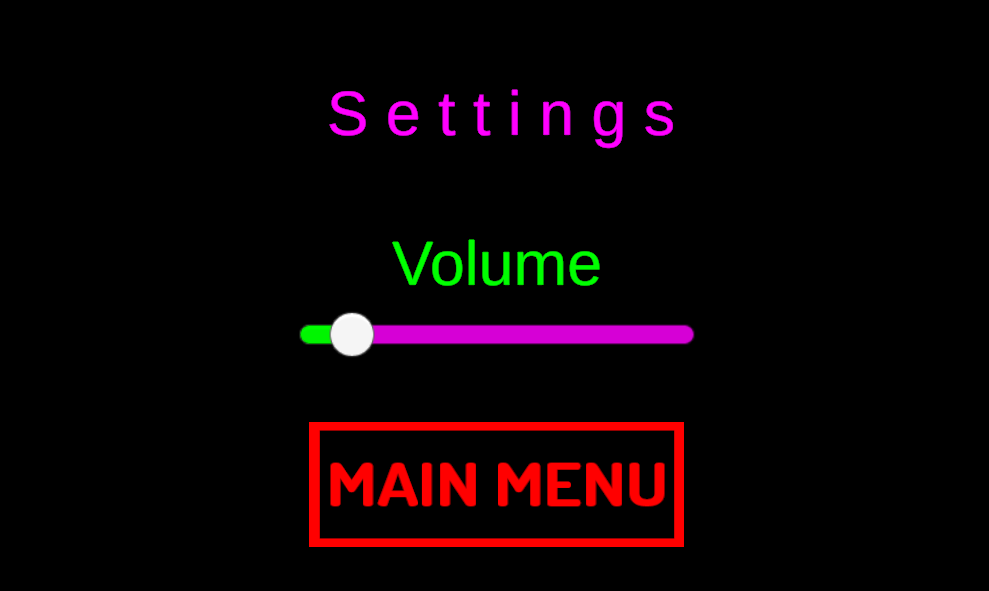


Рисунок 2.3 – Макет окна настроек веб-браузера

2.2.4 Окно завершения игры

Игра будет содержать окно завершения игры, которое будет открываться только в случае достижения необходимого количества забитых голов. В верхней части окна будет находится большая надпись «Game Over». Центральная часть окна будет представляется 2 кнопками «Replay», которая отвечает за перезапуск игрового процесса, и «Main Menu», которая отвечает за возвращение в главное меню.

Внешний вид информационных окон представлен на рисунке 2.4.



Рисунок 2.4 – Внешний вид информационных окон

## 2.3 Проектирование функционала программного средства

Граммотно поставленная задача и хорошо составленные алгоритмы – ключевая фаза в проектировании программного средства. Онлайн-игра Ping Pong Online должна обеспечивать минимально необходимый функционал для корректной и увлекательной игры между двумя участниками. Основные функции, которые должны быть реализованы в рамках игрового процесса:

– движение мяча с учётом столкновений с ракетками и границами игрового поля;

– управление ракеткой каждым игроком с синхронизацией через сервер;

– подсчёт очков и завершение игры при достижении победного результата.

Дополнительно необходимо обеспечить корректную передачу данных между клиентами через внешний сервер, а также стабильное отображение игрового состояния на экранах обоих игроков.

2.3.1 Движение мяча

Движение мяча осуществляется автоматически с момента его создания на сцене. После запуска игры мяч начинает движение по заранее заданному направлению с определённой скоростью. При столкновении с ракеткой или границей поля происходит перерасчёт направления движения, основанный на векторной физике отскока. Каждое обновление позиции мяча передаётся на сервер, который синхронизирует эти данные между клиентами.

Блок-схема метода, реализующего логику движения мяча, представлена на рисунке 2.6.

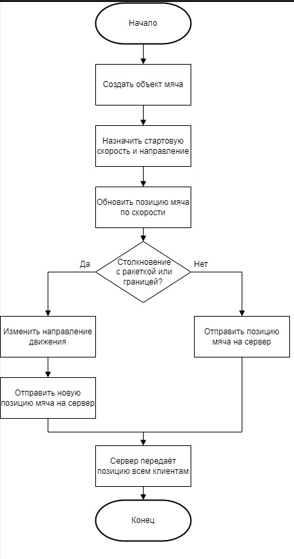


Рисунок 2.6 – Блок-схема метода UpdateBallPosition()

2.3.2 Управление ракеткой игрока

Управление ракеткой происходит при помощи нажатий клавиш «W» и «S» для игрока 1 (левая ракетка, если игрок «Host») для игрока 2 (правая ракетка, если игрок «Client»). При каждом нажатии соответствующей клавиши позиция ракетки обновляется и отправляется на сервер. Сервер, в свою очередь, рассылает обновлённые координаты другим клиентам, обеспечивая синхронное отображение.

Блок-схема обработки ввода игрока и синхронизации движения ракетки показана на рисунке 2.7.

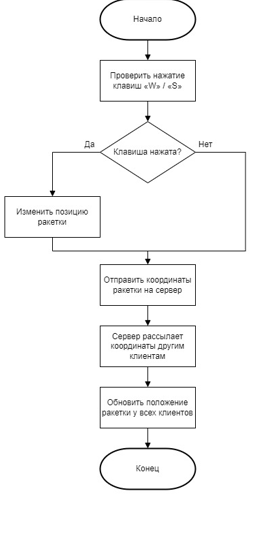


Рисунок 2.7 – Блок-схема метода InitializeRackets()

2.3.3 Подсчёт очков и завершение игры

Счёт обновляется при каждом случае, когда мяч касается левой или правой границы поля. В зависимости от стороны, игроку начисляется очко. После каждой потери мяча игра приостанавливается, и мяч создаётся заново. Когда один из игроков набирает 11 очков, срабатывает завершение матча и загружается сцена «GameOver». Игроки могут перезапустить игру, при этом соединение с сервером сохраняется, а счёт обнуляется.

Блок-схема логики подсчёта очков и обработки завершения игры представлена на рисунке 2.8.

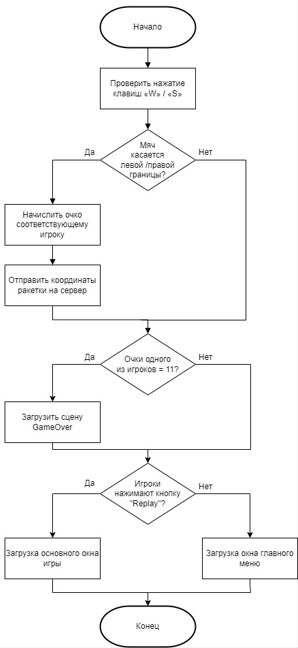


Рисунок 2.8 – Блок-схема метода AddScore()

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Движение мяча и управление игровым процессом

Ключевыми возможностями программного средства «Ping Pong Online» являются функции, связанные с движением игрового мяча и управлением взаимодействием между игроками. Движение мяча реализовано с учётом физики столкновений и синхронизируется через сервер, что позволяет обеспечить одинаковое отображение игрового состояния у всех подключённых клиентов.

В основе работы приложения лежит модуль «BallMovement», который отвечает за начальное направление движения мяча, его перемещение по сцене, а также логику изменения траектории при столкновениях с ракетками или границами поля. Отскоки реализованы с помощью компонента BallBounce, который определяет сторону контакта и пересчитывает вектор направления.

После каждого столкновения мяч продолжает движение с тем же темпом, что создаёт динамику игрового процесса. Все изменения координат мяча обрабатываются локально, но при этом отправляются на сервер, который пересылает актуальные данные всем клиентам. Это гарантирует, что положение мяча будет синхронным для обоих игроков.

Сброс мяча и начало нового розыгрыша происходят автоматически после каждого набранного очка. Инициализация мяча производится модулем BallManager, который создаёт объект мяча в центре сцены и задаёт начальный импульс для старта игры.

Таким образом, управление движением мяча, столкновениями и его повторным созданием представляет собой фундаментальную часть игрового цикла, от корректной работы которой зависит вся логика матча.

3.1.1 Управление мячом

Движение мяча в игре Ping Pong Online реализовано автоматически после запуска сцены. При старте игры мяч создаётся в центре поля, и ему задаётся случайное начальное направление. За это отвечает метод LaunchBall() из скрипта BallMovement.cs, вызываемый при старте или перезапуске мяча.

Если один из игроков пропускает мяч (т.е. мяч касается края поля), происходит сброс мяча: он возвращается в центр и снова запускается в произвольную сторону. Это реализовано в методе ResetBall(), который вызывается после получения соответствующего сигнала от сервера.

Сервер, в свою очередь, рассылает всем клиентам сообщение с координатами мяча в формате BALL:x:y, которое обрабатывается методом ProcessMessage() в скрипте NetworkClient.cs. Полученные координаты используются для обновления позиции мяча на стороне клиента.

Пример кода обработки движения мяча представлен ниже:

public void LaunchBall()

{

float dirX = Random.Range(0, 2) == 0 ? -1 : 1;

float dirY = Random.Range(-1f, 1f);

Vector2 direction = new Vector2(dirX, dirY).normalized;

rb.linearVelocity = direction \* speed;

}

Таким образом, взаимодействие компонентов «BallMovement», «BallManager» и «NetworkClient» обеспечивает реалистичное и синхронное поведение мяча на всех устройствах.

3.1.2 Управление ракетками игроков

Управление ракетками в игре Ping Pong Online осуществляется с помощью клавиатуры. Игрок 1 (если он запускает игру как Host) управляет левой ракеткой при помощи клавиш «W» (вверх) и «S» (вниз). Игрок 2 (если он Client) управляет правой ракеткой при помощи клавиш «стрелка вверх» и «стрелка вниз».

Каждое нажатие соответствующей клавиши приводит к перемещению ракетки вверх или вниз в пределах установленного вертикального диапазона. После перемещения координата Y ракетки отправляется на сервер в виде сообщения (P1:y или P2:y). Сервер пересылает полученные координаты второму клиенту, обеспечивая тем самым синхронное отображение позиций ракеток у обоих игроков.

Пример кода управления ракеткой и отправки координаты (Player1.cs):

if (NetworkManager.IsHost)

{

moveInput = Input.GetAxisRaw("Vertical");

Vector3 newPosition = transform.position + new Vector3(0, moveInput \* speed \* Time.deltaTime, 0);

newPosition.y = Mathf.Clamp(newPosition.y, -boundaryY, boundaryY);

transform.position = newPosition;

NetworkClient.Instance.Send($"P1:{transform.position.y:F2}");

}

Аналогичная логика реализована в Player2.cs для клиента, только с отправкой префикса P2.

После получения сообщения сервером оно обрабатывается и пересылается другому игроку, где вызывается метод UpdatePosition(), обновляющий позицию ракетки на экране:

public void UpdatePosition(float y)

{

transform.position = new Vector3(transform.position.x, y, transform.position.z);

}

Таким образом, точное и своевременное управление ракетками обеспечивается через клиент-серверное взаимодействие и постоянную синхронизацию координат, что позволяет добиться чёткого мультиплеерного игрового процесса.

## 3.2 Управление игровыми сценами и сессией

В игре Ping Pong Online важной составляющей является управление состояниями игры: от начального экрана до игрового процесса и экрана завершения. Вместо «вкладок» в классическом браузере, здесь используется система сцен Unity и UI-компонентов, которые позволяют плавно переключаться между различными этапами игры, управлять сессией и обеспечивать взаимодействие с сервером без потери соединения.

3.2.1 Запуск новой игровой сессии

Создание новой игровой сессии происходит по нажатию кнопки «Play» в главном меню. При этом загружается игровая сцена с помощью модуля ChangeScene, который использует метод SceneManager.LoadScene. При переходе автоматически инициируется подключение к серверу (через NetworkClient) и создаются игровые объекты — мячи, ракетки, UI-элементы.

Пример метода запуска сцены:

public void StartGame()

{

SceneManager.LoadScene("GameScene");

}

После загрузки сцены инициализируются компоненты BallManager, Player1, Player2, а также выполняется запрос к серверу о текущем состоянии игры (REQUEST\_INITIAL\_STATE), если клиент подключается к уже идущей сессии.

3.2.2 Завершение и перезапуск игры

При достижении одним из игроков 11 очков (значение хранится в ScoreManager.maxScore) происходит автоматическая загрузка сцены GameOver. За это отвечает модуль ScoreManager, который отслеживает счёт обоих игроков и вызывает SceneManager.LoadScene("GameOver"), если условие победы выполнено.

В сцене GameOver отображается итоговый счёт и кнопка «Replay», позволяющая начать новую сессию, не разрывая соединение с сервером. Для этого вызывается метод перезапуска, который сбрасывает счёт (ResetScores()), заново создаёт мяч (InitializeBall()) и перезапускает сцену GameScene.

Пример кода перезапуска:

public void RestartGame()

{

NetworkClient.Instance.ResetScores();

SceneManager.LoadScene("GameScene");

}

Таким образом, система сцен обеспечивает управление жизненным циклом игровой сессии.

**3.2.3** Переключение между экранами и отображение UI

Переключение между игровыми экранами реализовано с помощью стандартных средств Unity: SceneManager и UI-панелей. Например, во время паузы в игре активируется PauseMenu, который блокирует управление и отображает интерфейс с кнопками «Продолжить», «Выйти в меню» и т.д.

Управление этим интерфейсом осуществляется методом TogglePauseMenu() в классе GameManager, который включается по событию клавиши Escape или по сетевому сигналу PAUSE/UNPAUSE.

Пример:

public void TogglePauseMenu(bool isActive)

{

pauseMenuUI.SetActive(isActive);

Time.timeScale = isActive ? 0f : 1f;

}

Это позволяет гибко управлять игровыми экранами, отображать текущие состояния сессии и при необходимости приостанавливать матч.

## 3.3 Работа с игровыми событиями

В Ping Pong Online реализация игровых событий является важной частью механизма синхронизации состояния между клиентами. События обрабатываются на стороне клиента и передаются через сервер в виде сетевых сообщений. Такие события включают в себя: начисление очков, постановку игры на паузу, сброс счёта, завершение матча, переход в главное меню и др.

Обработка событий реализована в классе NetworkClient, где поступающие сообщения разбираются методом ProcessMessage() и соответствующим образом влияют на логику игры.

**3.3.1** Добавление игрового события

Игровые события добавляются в процессе взаимодействия игрока с игрой. Например, когда игрок пропускает мяч и соперник получает очко, клиент отправляет на сервер сообщение в формате SCORE:1:0 (если очко получил игрок 1) или SCORE:0:1 (если игрок 2). Это сообщение затем передаётся другим клиентам для обновления UI и проверки окончания игры.

Пример вызова события:

NetworkClient.Instance.Send($"SCORE:{player1Score}:{player2Score}");

Сообщение анализируется методом ProcessMessage(), где выполняется обновление локального счёта и, при необходимости, переход к сцене окончания игры.

Игровые события, подобно закладкам в браузере, позволяют «зафиксировать» определённое состояние игры и синхронизировать его между участниками.

**3.3.2** Удаление (сброс) игрового состояния

Сброс счёта осуществляется по команде RESET\_SCORES, которая передаётся клиентами на сервер, например, после завершения матча или при нажатии на кнопку «Replay». На стороне клиента она вызывает метод ResetScores():

public void ResetScores()

{

score1 = 0;

score2 = 0;

Send($"SCORE:{score1}:{score2}");

}

Сервер передаёт эти данные всем подключённым клиентам, вызывая у них локальное обнуление значений и, при необходимости, перезапуск игрового процесса. Таким образом, удаление игрового состояния выполняется централизованно и синхронно.

**3.3.3** Активация игрового состояния (открытие события)

При подключении нового клиента (в роли наблюдателя или второго игрока) требуется получение актуального состояния игры: текущего счёта и позиции мяча. Это достигается путём отправки сообщения REQUEST\_INITIAL\_STATE серверу. В ответ сервер отправляет клиенту события SCORE:x:y и BALL:x:y, чтобы воспроизвести текущее положение объектов. Пример обработки события:

case "SCORE":

score1 = int.Parse(parts[1]);

score2 = int.Parse(parts[2]);

// Обновление UI или проверка победителя

break;

case "BALL":

float x = float.Parse(parts[1]);

float y = float.Parse(parts[2]);

// Установка позиции мяча на клиенте

break;

Таким образом, при активации клиент получает все актуальные события. Это позволяет моментально восстановить текущее состояние игры и продолжить матч без рассинхронизации.

## 3.4 Работа с игровыми данными

Работа с игровыми данными в Ping Pong Online осуществляется в рамках игровой логики и сетевого взаимодействия. В основном это управление счётом игроков, положением мяча и позициями ракеток. Для управления игровыми данными используются классы ScoreManager, BallMovement, Player1, Player2 и NetworkClient.

В игре отображается информация о текущем счёте игроков через текстовые поля интерфейса (player1ScoreText и player2ScoreText в ScoreManager). Сброс игровых данных, таких как счёт и положение объектов, осуществляется через соответствующие методы этих классов. Для анализа и управления данными используются методы, такие как ResetScores() в ScoreManager и ResetBall() в BallMovement.

**3.4.1** Сброс положения мяча

Сброс положения мяча происходит после того, как один из игроков получает очко или игра перезапускается. Это реализовано в методе ResetBall() класса BallMovement. Код метода приведён ниже:

public void ResetBall()

{

rb.linearVelocity = Vector2.zero;

transform.position = Vector2.zero;

LaunchBall();

}

Метод сбрасывает скорость мяча (linearVelocity) до нуля, возвращает мяч в центр сцены (transform.position = Vector2.zero) и вызывает метод LaunchBall() для запуска мяча в случайном направлении с заданной скоростью. В сетевой игре хост отправляет обновлённое положение мяча клиентам через сообщение BALL:{x:F2}:{y:F2} в NetworkClient.

**3.4.2** Сброс позиций ракеток игроков

Сброс позиций ракеток игроков происходит через установку их координат по оси Y в начальное положение (обычно центр сцены). Это реализовано в методах UpdatePosition(float y) классов Player1 и Player2. Код метода UpdatePosition для Player1 приведён ниже (аналогичный код используется в Player2):

public void UpdatePosition(float y)

{

if (!NetworkManager.IsHost)

{

transform.position = new Vector3(transform.position.x, y, transform.position.z);

Debug.Log($"Клиент обновил позицию Player1: {y}");

}

}

Метод обновляет позицию ракетки по оси Y на основе полученного значения y, переданного через сетевое сообщение (P1:{y:F2} или P2:{y:F2}). Для сброса позиции ракетки в начальное положение (например, y = 0) хост или клиент отправляет соответствующее сообщение через NetworkClient. После этого игра может продолжиться с синхронизированными позициями ракеток.

**3.4.3** Сброс игровых данных в сетевом режиме

Сброс всех игровых данных в сетевом режиме (счёт, положение мяча и позиции ракеток) инициируется хостом и синхронизируется с клиентом через NetworkClient. Для этого используется метод ResetScores() в NetworkClient, который отправляет сообщение о сбросе счёта. Код метода приведён ниже:

public void ResetScores()

{

score1 = 0;

score2 = 0;

Send($"SCORE:{score1}:{score2}");

}

Метод сбрасывает локальные переменные score1 и score2 и отправляет сообщение SCORE:0:0 всем подключённым клиентам. Дополнительно хост может отправить сообщение BALL:{x:F2}:{y:F2} для сброса положения мяча и сообщения P1:{y:F2} и P2:{y:F2} для сброса позиций ракеток. После выполнения сброса игра перезапускается с начальными условиями, синхронизированными между хостом и клиентом.

**3.4.4** Синхронизация начального состояния игры

Синхронизация начального состояния игры необходима, когда клиент подключается к хосту, чтобы получить актуальные данные о счёте, положении мяча и ракеток. Это реализовано в NetworkClient через обработку сообщения REQUEST\_INITIAL\_STATE. Код обработки этого сообщения приведён ниже:

case "REQUEST\_INITIAL\_STATE":

if (NetworkManager.IsHost)

{

Debug.Log("Получен запрос на начальное состояние, отправляем данные...");

NetworkClient.Instance.Send($"SCORE:{score1}:{score2}");

if (BallManager.Instance != null && BallManager.Instance.ball == null)

{

BallManager.Instance.InitializeBall();

}

if (BallManager.Instance != null)

{

Rigidbody2D rb = BallManager.Instance.ball.GetComponent<Rigidbody2D>();

if (rb != null)

{

NetworkClient.Instance.Send($"BALL:{rb.position.x:F2}:{rb.position.y:F2}");

Debug.Log($"Хост отправил начальное положение мячика: ({rb.position.x:F2}, {rb.position.y:F2})");

}

else

{

Debug.LogError("Rigidbody2D не найден на мячике на хосте!");

}

}

}

break;

Когда клиент отправляет сообщение REQUEST\_INITIAL\_STATE, хост отвечает, отправляя текущий счёт через сообщение SCORE:{score1}:{score2} и положение мяча через сообщение BALL:{x:F2}:{y:F2}. Это обеспечивает синхронизацию игрового состояния между хостом и клиентом при подключении, гарантируя, что клиент начинает игру с актуальными данными.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения Ping Pong Online были выявлены некоторые ошибки и недочёты в работе сетевой игры. Основной недочёт — это рассинхронизация положения мяча между хостом и клиентом. Проблема была связана с тем, что клиент не всегда корректно обновлял положение мяча при получении сообщения BALL от хоста. Это происходило, если объект мяча (BallManager.Instance.ball) не был инициализирован на стороне клиента к моменту получения сообщения.

Для устранения этой проблемы в метод ProcessMessage класса NetworkClient была добавлена дополнительная проверка на наличие мяча и его инициализация при необходимости. Код обработки сообщения BALL в NetworkClient приведён ниже:

case "BALL":

float ballX = float.Parse(parts[1]);

float ballY = float.Parse(parts[2]);

Debug.Log($"Получено BALL: {ballX}, {ballY} на клиенте");

if (BallManager.Instance == null)

{

Debug.LogWarning("BallManager отсутствует в сцене! Убедитесь, что BallManager привязан к сцене Pong Game.");

return;

}

if (BallManager.Instance.ball == null)

{

Debug.LogWarning("Мячик не найден в BallManager! Инициализируем новый.");

BallManager.Instance.InitializeBall();

}

var ballScript = BallManager.Instance.ball.GetComponent<BallBounce>();

if (ballScript != null)

{

ballScript.UpdateBallPosition(ballX, ballY);

}

else

{

Debug.LogError("BallBounce не найден на мячике после инициализации!");

}

break;

Данный метод обрабатывает сетевое сообщение BALL, содержащее координаты мяча (ballX, ballY). Если объект BallManager отсутствует в сцене, выводится предупреждение в консоль. Если мяч (BallManager.Instance.ball) не инициализирован, вызывается метод InitializeBall() для его создания. После этого метод UpdateBallPosition обновляет позицию мяча на стороне клиента. Это гарантирует, что клиент всегда синхронизирован с хостом, даже если мяч не был создан на момент получения сообщения.

Дополнительно метод проверяет наличие компонента BallBounce на объекте мяча. Если компонент отсутствует, выводится сообщение об ошибке, что помогает выявить проблемы с настройкой объектов в сцене. Эта доработка устранила проблему рассинхронизации, обеспечивая корректное обновление положения мяча на стороне клиента.

Большинство проблем возникло из-за недочётов на стадии проектирования сетевой синхронизации, таких как недостаточная проверка инициализации игровых объектов. На стадии тестирования приложения все выявленные проблемы были исправлены.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 Главное окно

Главное окно приложения Ping Pong Online представляет собой стартовый экран игры. В верхней части окна отображается большая надпись с названием игры «Ping Pong Online». Ниже расположены кнопки для управления подключением и запуском игры: «Играть как хост», «Играть как клиент», «Настройки» и «Выход». Эти элементы позволяют пользователю выбрать роль (хост или клиент), открыть настройки или выйти из приложения. Внешний вид главного окна веб-браузера представлен на рисунке 5.1.

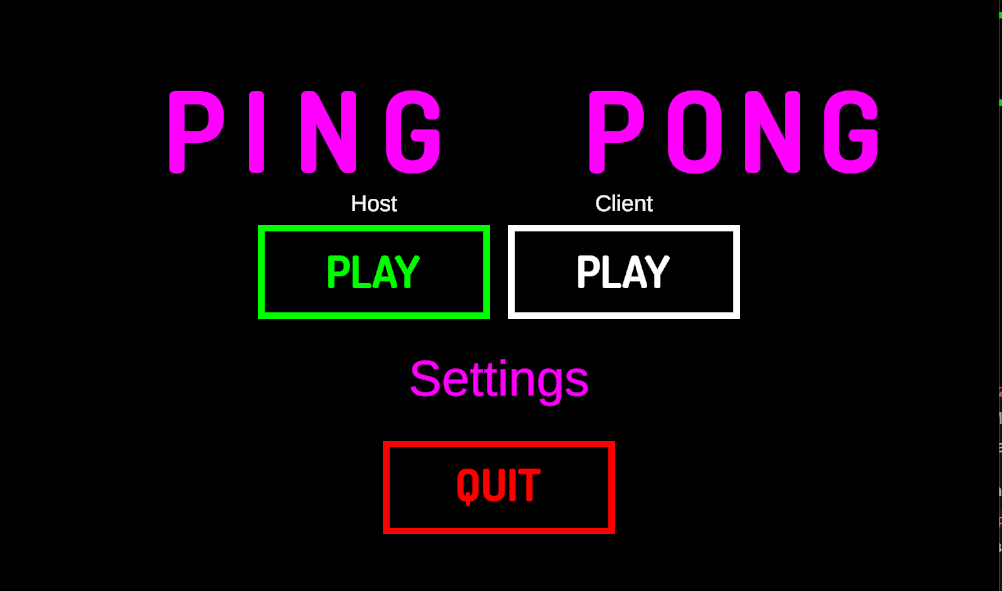


Рисунок 5.1 – Главное окно приложения

5.1.2 Окно настроек

Окно настроек состоит из двух частей. В верхней части окна отображается большая надпись «Settings». Центральная часть содержит элемент управления типа «Slider», который позволяет регулировать громкость музыки в игре. В нижней части окна расположена кнопка «Main Menu», возвращающая пользователя в главное меню. Внешний вид окна настроек представлен на рисунке 5.2.

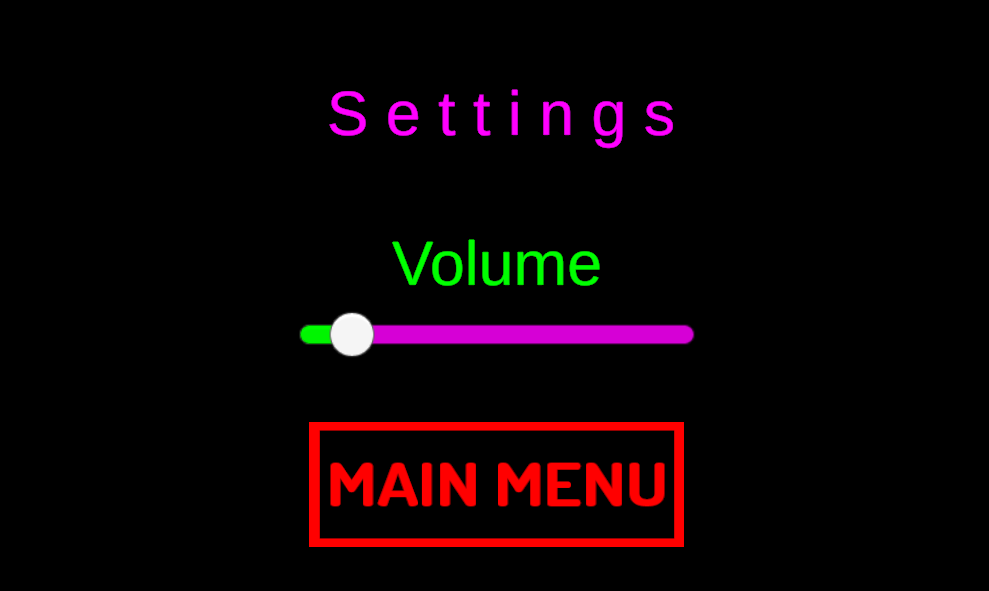


Рисунок 5.2 – Окно настроек

5.1.3 Окно завершения игры

В верхней части окна отображается надпись «Game Over». В центральной части расположены две кнопки: «Replay» для перезапуска игры и «Main Menu» для возврата в главное меню. Внешний вид окна завершения игры представлен на рисунке 5.3.



Рисунок 5.3 – Окно добавления закладок

5.1.4 Информационные окна

Основное окно игры содержит элементы игрового процесса: две ракетки игроков (Player1 и Player2), мяч (BallMovement), верхнюю и нижнюю границы для отскока мяча, счёт игроков и кнопку «Pause». Счёт отображается через текстовые поля player1ScoreText и player2ScoreText в ScoreManager. Кнопка «Pause» вызывает метод TogglePauseMenu в GameManager, который приостанавливает игру. Внешний вид основного окна игры представлен на рисунке 5.4.

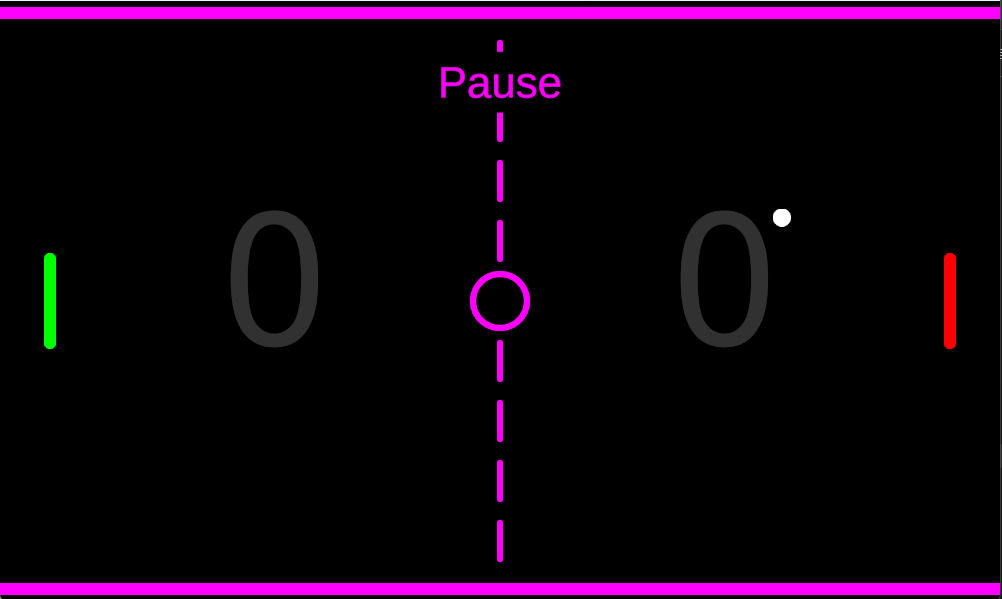


Рисунок 5.4 – Внешний вид информационных окон

## 5.2 Управление программным средством

5.2.1 Элементы управления игрой

Элементы управления игрой включают кнопки, клавиши ввода и текстовые поля для отображения счёта. При наведении на элементы интерфейса (например, кнопки в главном меню или окне настроек) могут отображаться всплывающие подсказки, описывающие их назначение (реализация зависит от настроек Unity UI). Пользователю доступны следующие функции:

– Запуск игры как хост: Нажатие кнопки «Играть как хост» в главном меню инициирует сервер и позволяет управлять ракеткой Player1 через клавиши ввода (Input.GetAxisRaw("Vertical") в Player1).

– Подключение как клиент: Нажатие кнопки «Играть как клиент» подключает игрока к хосту и позволяет управлять ракеткой Player2 через аналогичный ввод.

– Управление ракеткой: Движение ракетки вверх и вниз осуществляется с помощью клавиш (например, стрелок или WASD), обрабатываемых через Input.GetAxisRaw("Vertical") в Player1 и Player2

– Пауза игры: Нажатие кнопки «Pause» в основном окне игры отправляет сообщение PAUSE через NetworkClient, приостанавливая игру для всех игроков.

– Перезапуск игры: Нажатие кнопки «Replay» в окне завершения игры сбрасывает счёт и положение мяча, используя методы ResetScores() и ResetBall().

– Возврат в главное меню: Нажатие кнопки «Main Menu» в окнах настроек или завершения игры загружает сцену MainMenu.

– Отображение счёта: Счёт игроков обновляется в реальном времени через ScoreManager и синхронизируется через сообщения SCORE:{score1}:{score2}. Внешний вид элементов управления игрой представлен на рисунке 2.2.

5.2.2 Управление настройками игры

Управление настройками осуществляется в окне «Настройки». Пользователю доступны следующие возможности:

– Регулировка громкости музыки: Использование элемента «Slider» для изменения громкости (реализация предполагается через компоненты Unity, такие как AudioSource, хотя не отражена в предоставленном коде).

– Возврат в главное меню: Нажатие кнопки «Main Menu» загружает сцену главного меню через SceneManager.LoadScene("MainMenu"), как указано в обработке сообщения MAINMENU в NetworkClient.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была успешно спроектирована и реализована многопользовательская аркадная игра «Ping Pong Online» на игровом движке Unity. Основной задачей проекта являлась организация онлайн-взаимодействия между двумя игроками в реальном времени с помощью внешнего TCP-сервера. Данная задача была полностью решена.

В результате разработки:

– создан полноценный пользовательский интерфейс с возможностью запуска игрового клиента;

– реализована логика управления двумя ракетками с разделением управления по игрокам;

– обеспечена двусторонняя синхронизация позиций ракеток и мяча через сервер;

– реализован механизм учёта очков, визуальное отображение счёта и смена сцены при окончании игры;

– предусмотрена возможность перезапуска игры без потери соединения с сервером, что повышает стабильность и удобство;

– структурированы основные игровые классы и логика, что позволяет легко масштабировать проект в будущем.

Во время работы над проектом был получен ценный опыт в следующих направлениях:

– проектирование сетевой архитектуры с использованием TCP-соединений;

– управление многопоточностью и синхронизация данных между клиентом и сервером;

– работа с игровым движком Unity, включая управление сценами, объектами и UI-элементами;

– отладка и тестирование взаимодействия между несколькими экземплярами приложения.

Разработанный проект можно рассматривать как завершённый программный продукт, однако он обладает значительным потенциалом для расширения. В дальнейшем можно добавить поддержку игры по сети Интернет (через NAT и сервер матчмейкинга), реализовать регистрацию игроков, добавить звуковое сопровождение, а также визуальные улучшения (эффекты, анимацию, смену тем оформления). Возможно создание мобильной версии и внедрение веб-интерфейса на базе WebGL.

Таким образом, проект «Ping Pong Online» не только позволил продемонстрировать приобретённые навыки и знания, но и стал основой для дальнейшего развития в области сетевых приложений и разработки интерактивных игр с элементами реального взаимодействия между пользователями.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Holzner S. Unity 2022 Game Development Bible, 2022. — 1000 с.

[2] Thorn A. Unity Networking Fundamentals: Building Multiplayer Games,

2020. — 600 с.

[3] McConnell S. Code Complete: A Practical Handbook of Software Construction, 2nd Edition. — Microsoft Press, 2004. — 960 с.

[4] Алгоритмы. Теория и практические применение / Род Стивенс. —

Москва, 2016 — 544 с.

[5] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека

программиста. — СПб.: Питер, 2018. — 464 с.

[6] Lafore R. Object-Oriented Programming in C++, 4th Edition. — Sams Publishing, 2002. — 928 с.

[7] Nystrom R. Game Development with Unity and C#, 2021. — 512 с.

# ПРИЛОЖЕНИЕ А. Исходный код программы

«Server.cs»:

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

class Program

{

static TcpListener server;

static List<TcpClient> clients = new List<TcpClient>();

// Состояние игры

static float ballX = 0f, ballY = 0f;

static float player1Y = 0f, player2Y = 0f;

static int score1 = 0, score2 = 0;

static bool isPaused = false;

static void Main(string[] args)

{

// Запуск сервера на порту 7777

server = new TcpListener(IPAddress.Any, 7777);

server.Start();

Console.WriteLine("Сервер запущен на порту 7777. Ожидание подключений...");

// Поток для принятия новых клиентов

Thread acceptThread = new Thread(AcceptClients);

acceptThread.Start();

// Бесконечный цикл для поддержания сервера активным

while (true)

{

Console.ReadLine(); // Позволяет сервера ожидать ввода для завершения (можно заменить на другой механизм завершения)

}

}

static void AcceptClients()

{

while (true)

{

try

{

TcpClient client = server.AcceptTcpClient();

clients.Add(client);

Console.WriteLine($"Игрок подключён. Всего подключений: {clients.Count}");

// Запуск потока для обработки сообщений от клиента

Thread clientThread = new Thread(() => HandleClient(client));

clientThread.Start();

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при принятии клиента: {ex.Message}");

}

}

}

static void HandleClient(TcpClient client)

{

NetworkStream stream = client.GetStream();

byte[] buffer = new byte[1024];

while (client.Connected)

{

try

{

int bytesRead = stream.Read(buffer, 0, buffer.Length);

if (bytesRead == 0) break;

string message = Encoding.UTF8.GetString(buffer, 0, bytesRead).Trim();

Console.WriteLine($"Получено сообщение от клиента: {message}");

string[] parts = message.Split(':');

switch (parts[0])

{

case "BALL":

if (parts.Length == 3)

{

ballX = float.Parse(parts[1]);

ballY = float.Parse(parts[2]);

Broadcast($"BALL:{ballX:F2}:{ballY:F2}");

}

else

{

Console.WriteLine($"Неверный формат сообщения BALL: {message}");

}

break;

case "P1":

if (parts.Length == 2)

{

player1Y = float.Parse(parts[1]);

Broadcast($"P1:{player1Y:F2}");

Console.WriteLine($"Обработано P1: {player1Y:F2}");

}

else

{

Console.WriteLine($"Неверный формат сообщения P1: {message}");

}

break;

case "P2":

if (parts.Length == 2)

{

player2Y = float.Parse(parts[1]);

Broadcast($"P2:{player2Y:F2}");

Console.WriteLine($"Обработано P2: {player2Y:F2}");

}

else

{

Console.WriteLine($"Неверный формат сообщения P2: {message}");

}

break;

case "SCORE":

if (parts.Length == 3)

{

score1 = int.Parse(parts[1]);

score2 = int.Parse(parts[2]);

Broadcast($"SCORE:{score1}:{score2}");

}

break;

case "PAUSE":

isPaused = true;

Broadcast("PAUSE");

break;

case "UNPAUSE":

isPaused = false;

Broadcast("UNPAUSE");

break;

case "GAMEOVER":

Broadcast("GAMEOVER");

break;

case "RESET\_SCORES":

score1 = 0;

score2 = 0;

Broadcast("SCORE:0:0");

break;

case "MAINMENU":

Broadcast("MAINMENU");

break;

default:

Console.WriteLine($"Неизвестное сообщение: {message}");

break;

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка обработки клиента: {ex.Message}");

clients.Remove(client);

client.Close();

break;

}

}

Console.WriteLine("Клиент отключён. Осталось подключений: " + clients.Count);

clients.Remove(client);

client.Close();

}

static void Broadcast(string message)

{

byte[] data = Encoding.UTF8.GetBytes(message);

foreach (var c in clients.ToArray()) // Используем ToArray для избежания проблем с изменением коллекции

{

try

{

if (c.Connected)

{

NetworkStream stream = c.GetStream();

if (stream.CanWrite)

{

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

Console.WriteLine($"Отправлено всем: {message}");

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка отправки клиенту: {ex.Message}");

clients.Remove(c);

c.Close();

}

}

}

}

«NetworkClient.cs»:

using System;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using UnityEngine;

public class NetworkClient : MonoBehaviour

{

public static NetworkClient Instance { get; private set; }

private TcpClient client;

private NetworkStream stream;

public int score1 = 0;

public int score2 = 0;

public float player1Y = 0f;

public float player2Y = 0f;

private void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject);

Debug.Log("NetworkClient сохранён с DontDestroyOnLoad");

ConnectToServer();

}

else if (Instance != this)

{

Debug.LogWarning("Дубликат NetworkClient найден, уничтожаю...");

Destroy(gameObject);

return;

}

}

private void Start()

{

if (!IsConnected())

{

ConnectToServer();

}

}

public void ConnectToServer()

{

if (client == null || !client.Connected)

{

try

{

client = new TcpClient("127.0.0.1", 7777);

stream = client.GetStream();

Debug.Log("NetworkClient инициализирован и подключён к серверу.");

Thread receiveThread = new Thread(ReceiveMessages);

receiveThread.Start();

if (!NetworkManager.IsHost)

{

Send("REQUEST\_INITIAL\_STATE");

}

}

catch (Exception e)

{

Debug.LogError("Ошибка инициализации NetworkClient: " + e.Message);

}

}

}

public bool IsConnected()

{

return client != null && client.Connected && stream != null;

}

public void Send(string message)

{

if (stream == null || !stream.CanWrite)

{

Debug.LogWarning("Соединение не установлено. Переподключаюсь...");

ConnectToServer();

if (stream == null)

{

Debug.LogError("Не удалось восстановить соединение.");

return;

}

}

try

{

byte[] data = Encoding.UTF8.GetBytes(message + "\n");

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

Debug.Log($"Отправлено серверу: {message}");

}

catch (Exception e)

{

Debug.LogError($"Не удалось отправить сообщение: {e.Message}");

}

}

private void ReceiveMessages()

{

byte[] buffer = new byte[1024];

while (IsConnected())

{

try

{

int bytesRead = stream.Read(buffer, 0, buffer.Length);

if (bytesRead > 0)

{

string message = Encoding.UTF8.GetString(buffer, 0, bytesRead).Trim();

Debug.Log($"[Клиент] Получено от сервера: {message}");

ProcessMessage(message);

}

}

catch (Exception e)

{

Debug.LogError($"Ошибка при получении сообщения: {e.Message}");

break;

}

}

}

private void ProcessMessage(string message)

{

string[] parts = message.Split(':');

if (string.IsNullOrEmpty(message) || parts.Length < 1) return;

try

{

switch (parts[0])

{

case "P1":

if (parts.Length == 2 && float.TryParse(parts[1], out float player1YValue))

{

player1Y = player1YValue;

Debug.Log($"Получено P1: {player1YValue} (Роль: {(NetworkManager.IsHost ? "Хост" : "Клиент")})");

var player1 = GameObject.FindWithTag("Player1");

if (player1 != null)

{

var player1Script = player1.GetComponent<Player1>();

if (player1Script != null)

{

player1Script.UpdatePosition(player1YValue);

Debug.Log($"Обновлена позиция Player1 на: {player1YValue}");

}

}

}

break;

case "P2":

if (parts.Length == 2 && float.TryParse(parts[1], out float player2YValue))

{

player2Y = player2YValue;

Debug.Log($"Получено P2: {player2YValue} (Роль: {(NetworkManager.IsHost ? "Хост" : "Клиент")})");

var player2 = GameObject.FindWithTag("Player2");

if (player2 != null)

{

var player2Script = player2.GetComponent<Player2>();

if (player2Script != null)

{

player2Script.UpdatePosition(player2YValue);

Debug.Log($"Обновлена позиция Player2 на: {player2YValue}");

}

}

}

break;

case "BALL":

if (parts.Length >= 3)

{

float ballX = float.Parse(parts[1]);

float ballY = float.Parse(parts[2]);

Debug.Log($"[Клиент] Получено BALL: {ballX}, {ballY} (Роль: {(NetworkManager.IsHost ? "Хост" : "Клиент")})");

if (BallManager.Instance == null)

{

Debug.LogWarning("BallManager отсутствует в сцене!");

return;

}

if (BallManager.Instance.ball == null)

{

BallManager.Instance.InitializeBall();

Debug.Log("Клиент: Мяч инициализирован из-за отсутствия.");

}

var ballScript = BallManager.Instance.ball.GetComponent<BallBounce>();

if (ballScript != null)

{

ballScript.UpdateBallPosition(ballX, ballY);

Debug.Log($"Клиент: Вызван UpdateBallPosition с координатами ({ballX}, {ballY})");

}

else

{

Debug.LogError("BallBounce не найден на мячике!");

}

}

break;

case "SCORE":

if (parts.Length == 3 && int.TryParse(parts[1], out int newScore1) && int.TryParse(parts[2], out int newScore2))

{

score1 = newScore1;

score2 = newScore2;

Debug.Log($"Получено SCORE: {score1}, {score2}");

if (BallManager.Instance != null && BallManager.Instance.ball == null)

{

BallManager.Instance.InitializeBall();

}

if (BallManager.Instance != null)

{

var ballScriptScore = BallManager.Instance.ball.GetComponent<BallBounce>();

if (ballScriptScore != null)

{

ballScriptScore.UpdateScore(score1, score2);

}

}

}

break;

case "REQUEST\_INITIAL\_STATE":

if (NetworkManager.IsHost)

{

Debug.Log("Получен запрос на начальное состояние, отправляем данные...");

Send($"SCORE:{score1}:{score2}");

if (BallManager.Instance != null && BallManager.Instance.ball == null)

{

BallManager.Instance.InitializeBall();

}

if (BallManager.Instance != null && BallManager.Instance.ball != null)

{

Rigidbody2D rb = BallManager.Instance.ball.GetComponent<Rigidbody2D>();

if (rb != null)

{

Send($"BALL:{rb.position.x:F2}:{rb.position.y:F2}");

Debug.Log($"Хост отправил начальное положение мячика: ({rb.position.x:F2}, {rb.position.y:F2})");

}

}

}

break;

case "PAUSE":

Debug.Log("Получено PAUSE");

GameManager.Instance?.TogglePauseMenu(true);

break;

case "UNPAUSE":

Debug.Log("Получено UNPAUSE");

GameManager.Instance?.TogglePauseMenu(false);

if (!NetworkManager.IsHost)

{

Send("REQUEST\_INITIAL\_STATE");

}

break;

case "GAMEOVER":

Debug.Log("Получено GAMEOVER");

break;

case "RESET\_SCORES":

Debug.Log("Получено RESET\_SCORES");

score1 = 0;

score2 = 0;

GameManager.Instance?.ResetScores();

break;

case "MAINMENU":

Debug.Log("Получено MAINMENU");

UnityEngine.SceneManagement.SceneManager.LoadScene("MainMenu");

break;

default:

Debug.LogWarning($"Неизвестное сообщение: {message}");

break;

}

}

catch (Exception e)

{

Debug.LogError($"Ошибка парсинга сообщения '{message}': {e.Message}");

}

}

public void ResetScores()

{

score1 = 0;

score2 = 0;

Send($"SCORE:{score1}:{score2}");

}

private void OnDestroy()

{

if (client != null)

{

stream?.Close();

client.Close();

}

}

}

«BallBounce»:  
using UnityEngine;

using UnityEngine.SceneManagement;

public class BallBounce : MonoBehaviour

{

private Rigidbody2D rb;

private Vector2 lastVelocity;

private float speed = 5f;

private int score1 = 0;

private int score2 = 0;

private Vector2 targetPosition;

private Vector2 currentPosition;

private float lerpSpeed = 25f;

private bool hasReceivedPosition = false;

private void Awake()

{

rb = GetComponent<Rigidbody2D>();

currentPosition = Vector2.zero;

targetPosition = Vector2.zero;

Debug.Log($"BallBounce активирован: Ball на позиции: {transform.position}");

}

private void Start()

{

if (NetworkManager.IsHost)

{

ResetBall();

}

else

{

if (rb != null)

{

rb.isKinematic = true;

transform.position = Vector2.zero;

Debug.Log("Клиент: Физика мяча отключена, ждём позицию от хоста...");

}

}

}

private void Update()

{

if (NetworkManager.IsHost)

{

lastVelocity = rb.linearVelocity;

if (rb.linearVelocity.magnitude < 0.1f && SceneManager.GetActiveScene().name == "Pong Game")

{

Debug.LogWarning("Скорость мяча на хосте слишком мала, сбрасываем!");

ResetBall();

}

NetworkClient.Instance.Send($"BALL:{rb.position.x:F2}:{rb.position.y:F2}");

Debug.Log($"[Хост] Отправлено BALL: {rb.position.x:F2}, {rb.position.y:F2}");

}

else

{

if (hasReceivedPosition)

{

currentPosition = Vector2.Lerp(currentPosition, targetPosition, Time.deltaTime \* lerpSpeed);

transform.position = currentPosition;

Debug.Log($"[Клиент] Интерполяция мяча: текущая позиция: {currentPosition}, целевая: {targetPosition}");

}

else

{

Debug.Log("[Клиент] Ожидаем первую позицию мяча от хоста...");

}

}

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (!NetworkManager.IsHost) return;

if (collision.gameObject.CompareTag("TopWall") || collision.gameObject.CompareTag("BottomWall"))

{

Vector2 newVelocity = Vector2.Reflect(lastVelocity.normalized, collision.contacts[0].normal) \* speed;

rb.linearVelocity = newVelocity;

NetworkClient.Instance.Send($"BALL:{rb.position.x:F2}:{rb.position.y:F2}");

Debug.Log($"Мячик отскочил от {collision.gameObject.tag}. Новая скорость: {rb.linearVelocity}");

}

else if (collision.gameObject.CompareTag("Player1") || collision.gameObject.CompareTag("Player2"))

{

Vector2 newVelocity = Vector2.Reflect(lastVelocity.normalized, collision.contacts[0].normal) \* speed;

rb.linearVelocity = newVelocity;

NetworkClient.Instance.Send($"BALL:{rb.position.x:F2}:{rb.position.y:F2}");

Debug.Log($"Мячик отскочил от {collision.gameObject.tag}. Новая скорость: {rb.linearVelocity}");

}

else if (collision.gameObject.CompareTag("LeftWall") || collision.gameObject.CompareTag("RightWall"))

{

if (collision.gameObject.CompareTag("LeftWall"))

{

score2++;

Debug.Log($"Счёт увеличен для Player2: {score2}");

}

else if (collision.gameObject.CompareTag("RightWall"))

{

score1++;

Debug.Log($"Счёт увеличен для Player1: {score1}");

}

NetworkClient.Instance.Send($"SCORE:{score1}:{score2}");

ResetBall();

}

}

public void ResetBall()

{

if (!NetworkManager.IsHost) return;

transform.position = Vector2.zero;

rb.linearVelocity = Vector2.zero;

rb.angularVelocity = 0f;

float randomAngle = Random.Range(-45f, 45f);

Vector2 direction = new Vector2(Random.Range(0.7f, 1f) \* (Random.value > 0.5f ? 1 : -1), Mathf.Tan(randomAngle \* Mathf.Deg2Rad)).normalized;

rb.linearVelocity = direction \* speed;

Debug.Log($"Мячик сброшен. Начальная скорость: {rb.linearVelocity}");

NetworkClient.Instance.Send($"BALL:{transform.position.x:F2}:{transform.position.y:F2}");

}

public void UpdateBallPosition(float x, float y)

{

if (NetworkManager.IsHost) return;

targetPosition = new Vector2(x, y);

if (!hasReceivedPosition)

{

currentPosition = targetPosition;

transform.position = currentPosition;

hasReceivedPosition = true;

Debug.Log($"[Клиент] Первая позиция мяча получена: {targetPosition}, hasReceivedPosition: {hasReceivedPosition}");

}

Debug.Log($"[Клиент] Обновлена целевая позиция мяча: {targetPosition}");

}

public void UpdateScore(int newScore1, int newScore2)

{

score1 = newScore1;

score2 = newScore2;

Debug.Log($"Обновлён счёт: Player1 = {score1}, Player2 = {score2}");

if (GameManager.Instance != null)

{

GameManager.Instance.UpdateScoreDisplay(score1, score2);

}

}

public void GetScore(out int s1, out int s2)

{

s1 = score1;

s2 = score2;

}

}