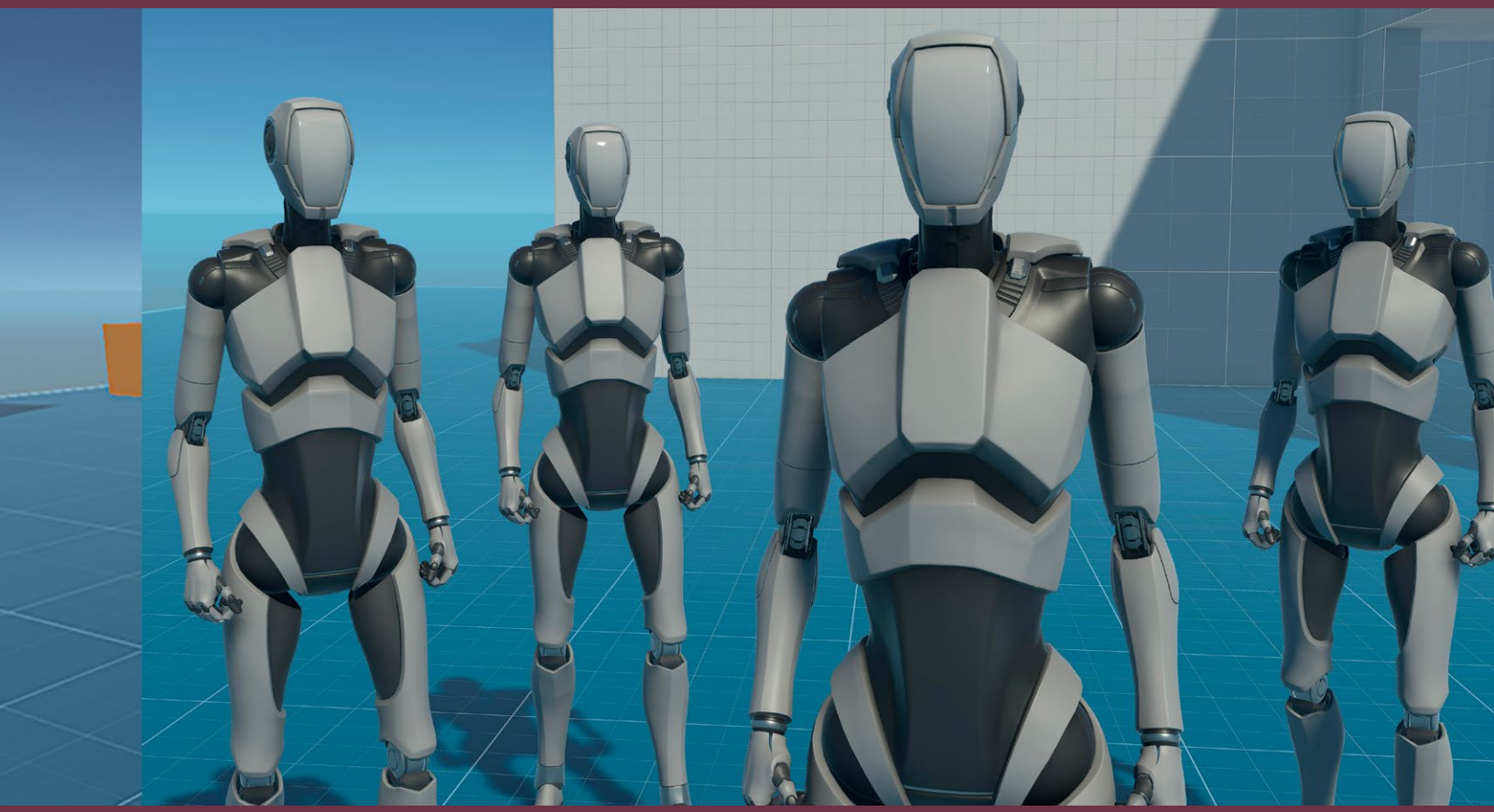
ЭЛЕКТРОННАЯ КНИГА

Введение в сетевое взаимодействие многопользовательских игр в Unity



Unity®

Содержание

Введение

Когда люди встречаются в онлайн-игре, происходит нечто особенное. Обычная гонка или ролевая игра превращается в совместное приключение - непредсказуемое, непростое и, самое главное, увлекательное

Объединяемся ли мы против межзвездного диверсанта или сражаемся друг с другом в онлайн-шутере, сетевые многопользовательские игры позволяют сотрудничать и соревноваться независимо от расстояния. Именно такое совместное взаимодействие определяет многопользовательский игровой опыт

Однако разработка сетевого игрового процесса может быть сложнее создания сопоставимой однопользовательской игры

Цель этого руководства - помочь вам начать разработку многопользовательских игр с помощью сетевых инструментов Unity. Оно знакомит с основными понятиями сетевого взаимодействия и подготавливает к работе с демонстрационными сетевыми проектами Unity. Кроме того, в нем пошагово разобран базовый пример на основе пакета Starter Assets - ThirdPerson из Unity Asset Store

Предполагается, что вы знакомы с Unity и разработкой на C#, но только начинаете осваивать сетевое взаимодействие. Руководство поможет быстро разобраться в теоретических основах многопользовательской разработки и подготовиться к практическим примерам

В этой электронной книге вы:

- Изучите ключевые понятия многопользовательской разработки в Unity.

- Познакомитесь с многопользовательскими системами и сетевыми моделями.

- Настроите простой пример с использованием Netcode for GameObjects.

Эволюция инструментов Unity для многопользовательских игр

Unity предлагает широкий набор инструментов и решений для многопользовательской разработки. Среди них - фреймворки Netcode for GameObjects и Netcode for Entities, а также Unity Gaming Services (UGS), куда входят Game Server Hosting (Multiplay) и Vivox для голосового и текстового общения. Создаете ли вы казуальную кооперативную игру или MMO с открытым миром, Unity поможет быстро приступить к разработке.

Unity 6 предлагает новые и улучшенные возможности для многопользовательских игр, благодаря которым интеграция, итерации и развертывание стали быстрее и надежнее. Кратко рассмотрим основные нововведения Unity 6 для многопользовательской разработки:

- Multiplayer Center: этот основной пакет Unity 6 упрощает настройку и разработку многопользовательских игр. Новые подсказки и рабочие процессы учитывают параметры и требования проекта, рекомендуют подходящие пакеты и сервисы, а затем создают динамические шаблоны для начала работы.

- Multiplayer Widgets: эти виджеты помогают интегрировать Unity Gaming Services (UGS) в многопользовательскую игру. Они доступны как отдельный пакет или через Multiplayer Center.

- Multiplayer Tools: пакет совершенствует рабочие процессы многопользовательской разработки в Unity, повышает производительность Netcode for GameObjects 2.0 и добавляет поддержку Distributed Authority.

- Multiplayer Play Mode: пакет упрощает проверку игрового процесса, позволяя запускать до четырех независимых облегченных процессов Editor с общим набором ресурсов на диске. Для наиболее масштабных проектов с серверным хостингом Play Mode Scenarios позволяет

настроить этапы развертывания, включая сборку выделенного сервера и его непосредственную загрузку на серверы Multiplay Hosting.

- Distributed Authority (beta): по состоянию на ноябрь 2024 года пакет Distributed Authority для Netcode for GameObjects находился на стадии beta. Он распределяет между клиентами полномочия владения созданными во время сеанса объектами NetworkObject. Нагрузка сетевой симуляции распределяется по клиентам, а состояние сети координирует высокопроизводительная облачная серверная часть Unity.

- Multiplayer Services SDK: единое решение для добавления многопользовательских возможностей в игру на Unity 6. Сервисы Unity Gaming Services (UGS), включая Relay и Matchmaker, обеспечивают работу Sessions, через которые задается взаимодействие групп игроков. Sessions совместимы с сетевыми библиотеками Netcode for GameObjects и Netcode for Entities.

Чтобы больше узнать о многопользовательских решениях Unity 6, ознакомьтесь со следующими материалами:

- Unite 2024: ускорение разработки соревновательной многопользовательской игры

- Unite 2024: переход к многопользовательской игре - как помочь студии и проекту добиться успеха

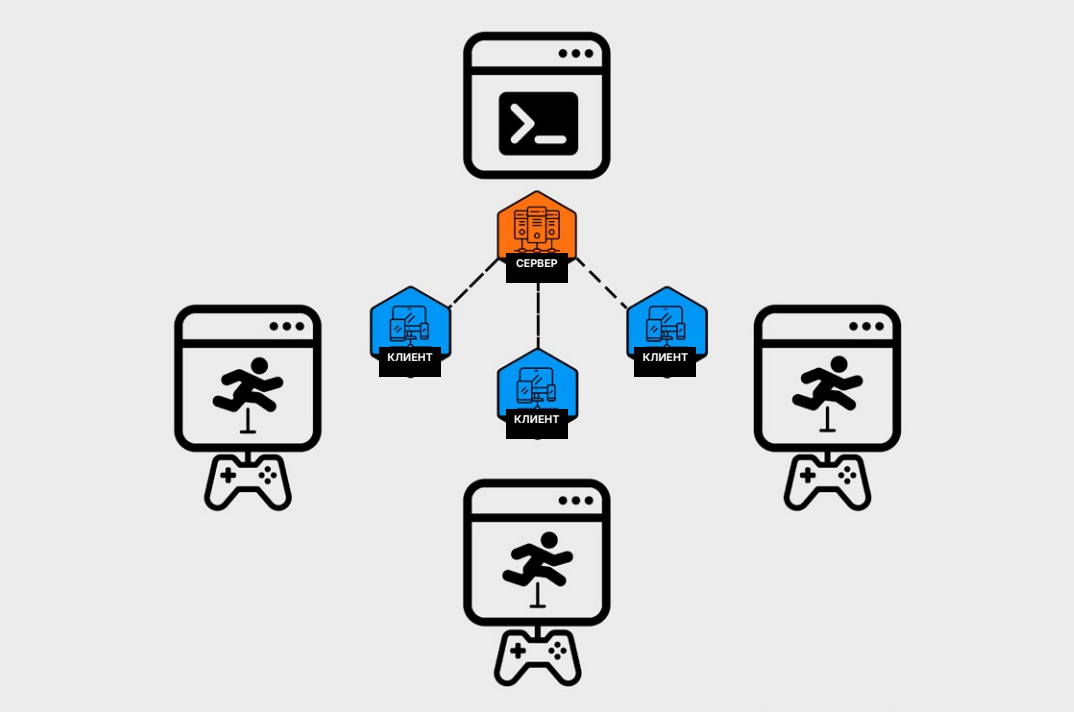
- Unity Manual: нововведения для многопользовательских игр в Unity 6

- Unity 6 уже здесь: обзор нововведений

Набор инструментов Unity охватывает весь жизненный цикл многопользовательской игры - от концепции и прототипа до выпуска и дальнейшей эксплуатации. Можно работать целиком в экосистеме Unity или выбрать только те инструменты и сервисы, которые лучше всего отвечают потребностям команды.

Основные понятия

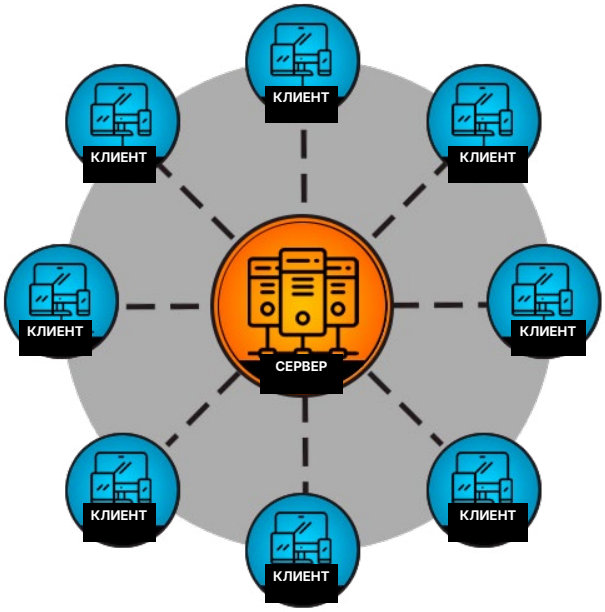
В многопользовательских играх сетевое взаимодействие позволяет игрокам подключаться к центральному серверу или непосредственно друг к другу, обмениваться данными и играть вместе в реальном времени. На устройстве каждого игрока одновременно работает один и тот же экземпляр игры, а действия игроков синхронизируются по сети.



В сетевой многопользовательской игре одно и то же приложение работает одновременно на нескольких устройствах.

Архитектура типичной сетевой игры включает два основных компонента: клиенты и серверы. Клиенты - это экземпляры игры, запущенные на устройствах игроков: компьютерах, консолях или мобильных телефонах. Они выводят графику, воспроизводят звук, обрабатывают пользовательский ввод и сообщают серверу о действиях игрока.

Серверы, в свою очередь, управляют состоянием игры - актуальными данными обо всех ее элементах. Они обеспечивают обмен между клиентами и следят за соблюдением игровых правил. Сервер может быть выделенным процессом без графического интерфейса, которым управляет разработчик, либо работать на устройстве одного из игроков, одновременно выступающего хостом.



Клиент-серверная архитектура

Сервер получает ввод от клиентов, обрабатывает игровую логику и отправляет обновления обратно. Хотя окончательные полномочия над состоянием игры принадлежат серверу, клиенты поддерживают локальные копии данных, чтобы игровой процесс оставался отзывчивым. Постоянная синхронизация обеспечивает максимально согласованное состояние игры для всех участников в реальном времени.

Серверы без графического интерфейса

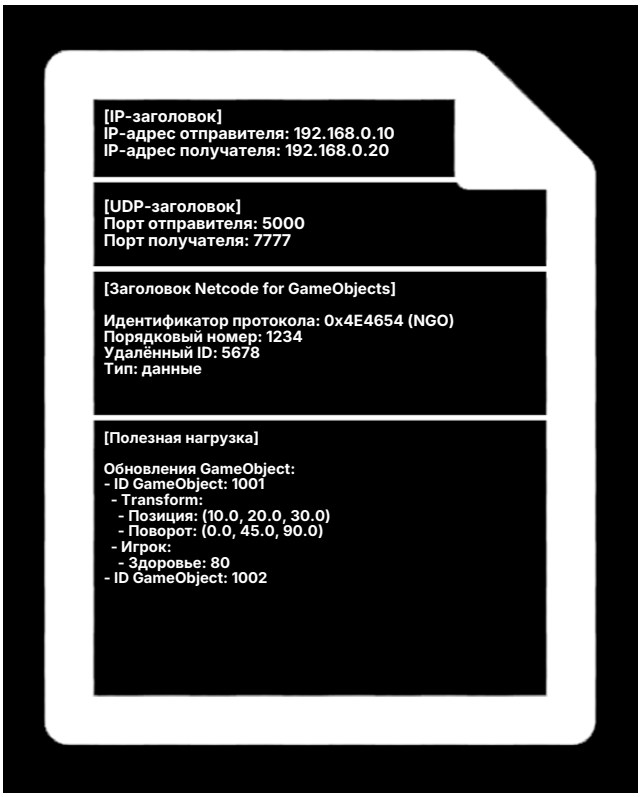
Сервер без графического интерфейса - это сервер, который работает без графического пользовательского интерфейса и выполняет исключительно серверные задачи.

Поскольку такие серверы не отрисовывают графику, их проще масштабировать; обычно их развертывают на выделенном оборудовании или в облачной среде. Подробнее см. раздел «Выделенный игровой сервер» в главе о сетевых топологиях.

Пакеты UDP

Клиенты и серверы взаимодействуют, обмениваясь пакетами данных по стандартным интернет-протоколам, например UDP (User Datagram Protocol - протокол пользовательских дейтаграмм). В приложениях реального времени, особенно в динамичных играх вроде шутеров от первого лица, UDP обычно предпочитают TCP (Transmission Control Protocol - протокол управления передачей), поскольку он позволяет самой игре определять приоритеты различных видов трафика. В отличие от TCP, UDP не требует подтверждения доставки от получателя. Благодаря этому UDP служит более эффективной и гибкой основой, но необходимые механизмы надежности приложение должно реализовать самостоятельно.

Каждый UDP-пакет состоит из заголовка и полезной нагрузки. Полезная нагрузка UTP содержит дополнительные разделы конкретного протокола, у каждого из которых есть собственные заголовок и полезная нагрузка. В сетевой терминологии такая вложенность называется инкапсуляцией.



Упрощенная структура UDP-пакета

Заголовки IP и UDP имеют фиксированный размер и содержат важные метаданные, в том числе адреса отправителя и получателя - IP-адреса и порты. Размер, структура и содержимое полезной нагрузки зависят от конкретной игры и ситуации. Например, в ней могут передаваться команды игрока или снимок состояния игры в определенный момент.

UDP-пакеты обеспечивают высокую скорость и низкую задержку, необходимые приложениям реального времени, но платой за это становится отсутствие встроенной надежности. UDP не предоставляет механизмов гарантированной доставки, упорядочивания пакетов или управления перегрузкой. При передаче через интернет возможны следующие проблемы:

- Потеря пакета: пакет может потеряться и не достичь получателя из-за перегрузки сети, неисправного оборудования или других причин.

- Дублирование: один пакет может быть доставлен получателю несколько раз, например из-за неправильной настройки сетевого оборудования или программного обеспечения.

- Нарушение порядка: пакеты могут прибыть в ином порядке, чем были отправлены. Это происходит, например, когда они следуют к получателю разными маршрутами с неодинаковой задержкой.

- Повреждение: содержимое пакета может измениться при передаче, и данные станут непригодными для использования.

Сравнение UDP и TCP

Сетевой протокол - это набор правил и соглашений, которые определяют, как данные передаются и принимаются по сети. Протоколы определяют, как устанавливать соединения, форматировать сообщения, обрабатывать ошибки и передавать данные.

При разработке игр UDP обычно предпочитают Transmission Control Protocol (TCP) благодаря его скорости и небольшим накладным расходам. TCP надежен и хорошо подходит, например, для просмотра веб-страниц: он обеспечивает доставку данных в правильном порядке, повторно отправляя потерянные пакеты и пакеты, пришедшие не по порядку. Однако это может вызывать задержки и рывки, неприемлемые для игр реального времени.

UDP, напротив, допускает некоторую потерю данных ради производительности в реальном времени. Поэтому игра может плавно работать с частотой 60 кадров/с и выше, не замирая при потере части некритичных данных. Обычно UDP обеспечивает удачный баланс между отзывчивостью и редкими потерями, поэтому хорошо подходит для сетевых многопользовательских игр.

Следующие приемы помогают компенсировать ненадежность UDP. В отличие от встроенных механизмов TCP, их можно точно настроить с учетом требований игры реального времени.

Механизм	Назначение
Порядковые номера	Каждому пакету присваивается уникальный возрастающий номер. Получатель обнаруживает пропущенные пакеты и нарушения порядка.
Подтверждения (ACK) и битовые маски ACK	Пакет содержит номер последнего полученного пакета. Битовая маска ACK показывает состояние нескольких пакетов и ускоряет обнаружение потерь.
Настройка тайм-аута повторной передачи (RTO)	Оценка времени кругового пути помогает настраивать интерполяцию и прогнозирование на стороне клиента.
Тайм-ауты	Если подтверждение не получено за заданное время, пакет считается потерянным.

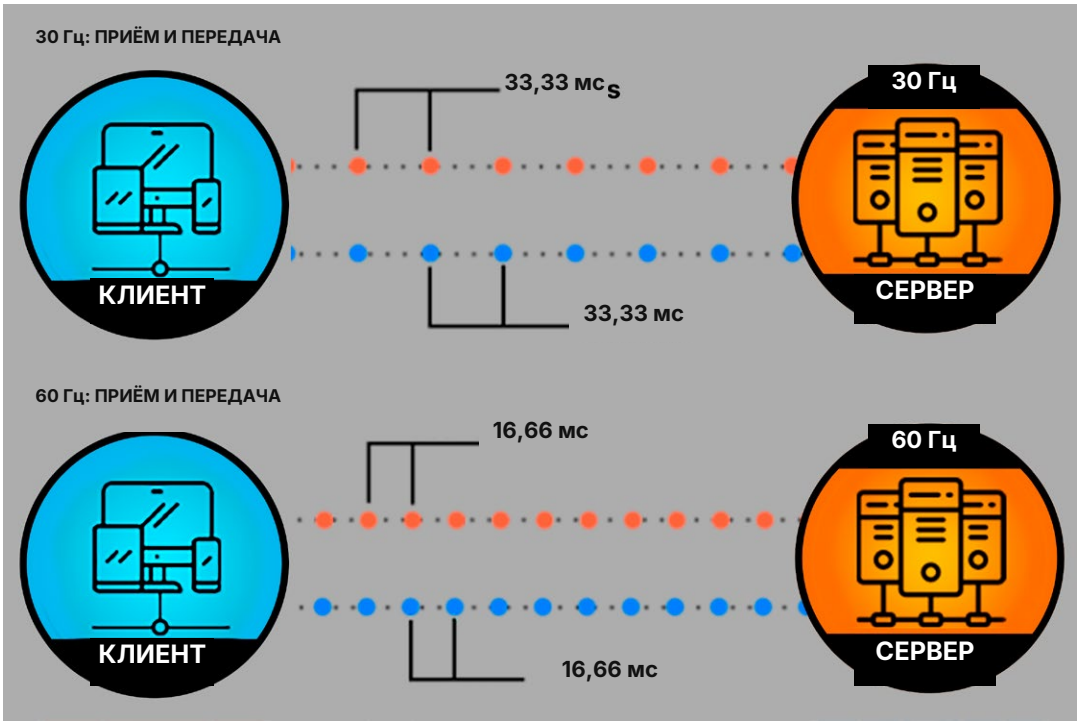
Тики и обновления

В многопользовательской игре сервер обрабатывает основную игровую логику, физическую симуляцию и другие функции игрового процесса, даже если работает без дисплея.

Сервер поддерживает глобальное состояние игры и получает ввод от клиентов, подобно тому как однопользовательская игра получает команды локального игрока. Только вместо собственной мыши и клавиатуры сервер обрабатывает присланные клиентами данные и на их основе поддерживает авторитетное состояние игры: позиции игроков, состояния объектов, результаты физических расчетов, прогресс и другие сведения.

Основа серверной обработки - серверный тик. Это цикл, в течение которого сервер обновляет состояние игры по полученным входным данным. Циклы выполняются с фиксированным интервалом, а их частота называется частотой тиков и измеряется в герцах (Гц), то есть тиках в секунду. Она определяет, как часто обновляется игровой мир.

Частота обновлений, в свою очередь, показывает, как часто клиент обменивается данными с сервером. Ее увеличение может сделать игру отзывчивее, но потребует большей пропускной способности и вычислительных ресурсов. Обычно частоту ограничивают возможности сети и оборудования клиента.



Частота тиков и частота обновлений

Более высокая частота тиков может повысить отзывчивость игры, но увеличивает нагрузку на сервер. Аналогично, повышение частоты обновлений делает взаимодействие плавнее ценой роста сетевого трафика. Чтобы многопользовательская игра оставалась плавной и отзывчивой, необходимо найти

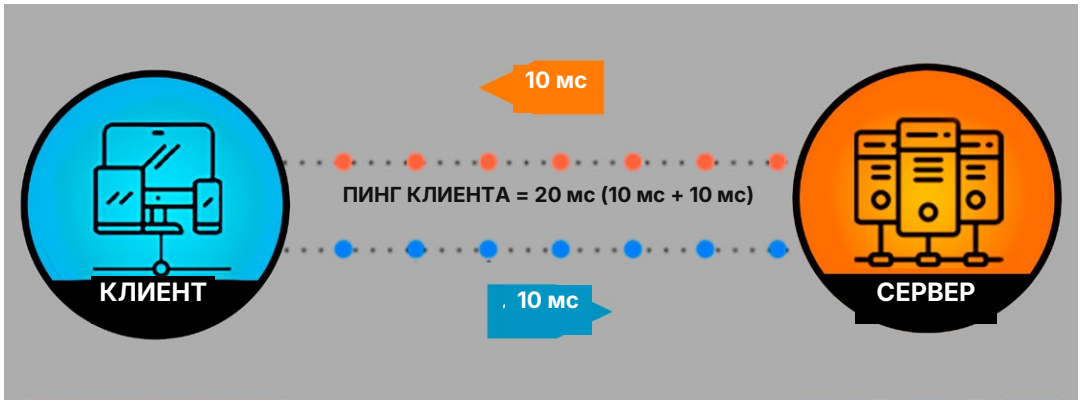
правильный баланс между частотой тиков и частотой обновлений.

Требуемая частота тиков зависит от игрового процесса. Динамичный шутер от первого лица часто работает с частотой 60 Гц и выше, чтобы точно отражать быстрые перемещения и мгновенные выстрелы. Стратегия в реальном времени меньше зависит от скорости реакции, поэтому ей может быть достаточно 30 Гц. А масштабная стратегическая MMO способна использовать сравнительно низкую частоту 10 Гц, чтобы обслуживать множество одновременно подключенных игроков.

Задержка

Задержка - это время, за которое данные проходят от источника до получателя. Время приема-передачи (round-trip time, RTT) показывает, сколько занимает путь пакета до адресата и возвращение ответа, и служит показателем сетевой задержки.

Оценка субъективна, но обычно ухудшение игрового процесса становится заметным при задержке около 200 мс. Допустимый уровень зависит от жанра: шутеры от первого лица лучше всего работают при задержке менее 100 мс, тогда как стратегия в реальном времени может оставаться приемлемой при значениях до 500 мс.



Время приема-передачи служит показателем сетевой задержки.

Низкая задержка делает управление отзывчивым: в идеале между действием игрока и ожидаемым результатом проходит минимум времени. При высокой задержке паузы в игровом процессе становятся заметными.

Причина задержки не всегда связана с сетью. Ее могут вызвать позднее распознавание ввода или сбой в конвейере рендеринга. Еще один источник - VSync: эта функция устраняет разрывы изображения, но добавляет задержку.

Однако чаще всего главным источником задержки остается сама сеть. Различают несколько ее составляющих:

- Задержка обработки: маршрутизатору требуется время, чтобы прочитать заголовок пакета и переслать его дальше. Обычно эта задержка невелика, но накапливается при прохождении нескольких сетевых узлов.

- Задержка передачи: время, необходимое для помещения пакета в сеть; оно напрямую зависит от размера пакета. Особенно заметно в сетях конечных пользователей с невысокой пропускной способностью.

- Задержка в очереди: при перегрузке или ограниченной пропускной способности интерфейса пакеты ожидают отправки в очереди, что может существенно увеличить общее время доставки.

- Задержка распространения: сигналу требуется время, чтобы пройти по сети. Эта задержка прежде всего зависит от физического расстояния между сервером и пользователем, среды передачи - оптоволокна, медного кабеля или воздуха - и вида сигнала: электрического, оптического или радиосигнала.

Полностью устранить задержку невозможно, особенно в интернет-играх, но ее влияние можно уменьшить с помощью упреждения, прогнозирования, интерполяции и других методов. Некоторые из них мы рассмотрим далее.

Другие сетевые термины

Ниже приведены еще несколько терминов, которые встречаются при обсуждении задержки:

Ping: отправка простого сообщения и получение ответа для оценки отзывчивости сети. Это упрощенный вариант измерения времени приема-передачи (RTT).

Jitter: колебания RTT из-за изменения состояния сети. Они могут снизить эффективность компенсации задержки и привести к нарушению порядка доставки пакетов.

Пропускная способность: объем данных, который сеть может передать за единицу времени. Высокая пропускная способность особенно важна для игр, пересылающих большие объемы данных о состоянии.

Эти и другие определения приведены в глоссарии Multiplayer Networking Terminology.

Сетевая синхронизация

Для синхронизации клиенты и сервер постоянно обмениваются сообщениями, поддерживая согласованное состояние игры у всех игроков. Обычно клиенты часто отправляют серверу команды пользователя - например, с частотой 60 Гц, то есть примерно каждые 16 мс. Это могут быть действия или входные данные: движение мыши или стика геймпада, а также нажатия кнопок прыжка и стрельбы.

Получив и обработав команды клиентов, сервер отправляет им обновленное состояние игрового мира. Чем быстрее игра реагирует на ввод игрока, тем отзывчивее воспринимается управление.

Важно помнить, что частота тиков сервера, частота обновлений клиента и частота кадров клиента решают разные задачи и не обязаны совпадать. Идеальная синхронизация на практике встречается редко: сервер и клиенты непрерывно меняют состояние и обмениваются потоком динамических данных. Цель состоит в том, чтобы свести расхождения к минимуму и создать впечатление, будто все участники играют в едином мире.

Методы сетевой синхронизации

Синхронизация состояния предполагает, что сервер периодически передает клиентам состояние сетевых объектов. Частота таких обновлений зависит от потребностей жанра - например, у соревновательного шутера и кооперативной стратегии они различаются.

Удаленные вызовы процедур (RPC) позволяют вызывать функции на сервере или других клиентах. Их используют для обмена между клиентом и сервером: отправки команд игрока, запроса конкретного действия или запуска однократного игрового события.

Грамотное управление пропускной способностью существенно влияет на производительность. Синхронизация расходует сетевой ресурс, поэтому объем передаваемых данных следует сокращать, например с помощью следующих приемов:

- Отсечение данных: исключение необязательных обновлений из сетевого трафика, чтобы передавать только сведения, необходимые для игрового процесса. Например, можно синхронизировать лишь критически важные оси перемещения, а визуальные эффекты и анимацию запускать локально по событиям вместо их непрерывной синхронизации. Любое сокращение трафика способно повысить производительность игры.

- Дельта-сжатие, или дельта-кодирование: сервер передает только изменения - дельты - относительно предыдущего обновления. Клиенты применяют эти дельты к локальному состоянию игры и тем самым поддерживают его синхронизацию с сервером.

- Управление областью интереса: приоритет синхронизации определяется по нескольким критериям. Пространственная релевантность учитывает расстояние до игрока и видимость объекта. Возраст, или степень устаревания, повышает приоритет данных, которые давно не передавались. Взаимодействие отдает предпочтение объектам, недавно взаимодействовавшим с игроком или способным сделать это в ближайшее время.

Эти методы помогают оптимизировать сетевую производительность и сделать игровой процесс более плавным и эффективным.

Топологии сети

Проще говоря, сетевая топология определяет, как устройства соединяются и обмениваются данными в многопользовательской среде. У каждой модели есть преимущества и недостатки. Выбор зависит от жанра игры, требуемого уровня контроля над ее состоянием и ресурсов, доступных для серверной инфраструктуры.

Топология влияет на архитектуру и производительность игры, а также на впечатления игроков. Netcode for GameObjects поддерживает две основные топологии: клиент-сервер и Distributed Authority. Разберемся, что это означает.

Клиент-серверная топология

Клиент-серверная топология - распространенная сетевая модель, которая распределяет обязанности между клиентскими устройствами и центральным сервером, чтобы эффективно управлять игрой и оптимизировать производительность.

Клиент представляет экземпляр игры конкретного игрока и обрабатывает локальный ввод, рендеринг и часть симуляции состояния. Клиенты отправляют серверу команды, например перемещение персонажа, и получают от него обновления.

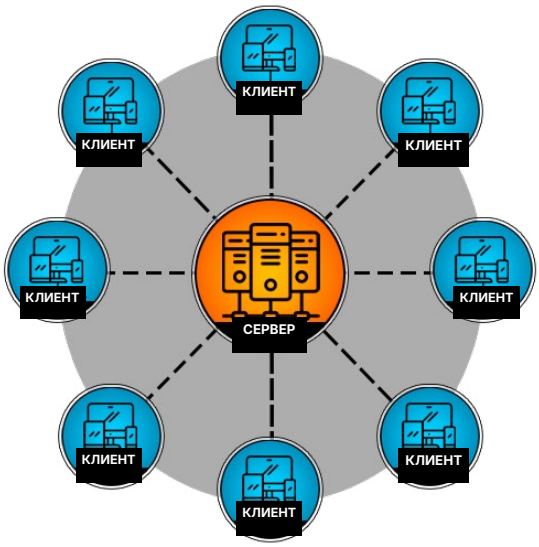
Сервер хранит окончательное и достоверное представление игрового мира, обрабатывает ввод игроков и обеспечивает соблюдение правил. Он разрешает конфликты и проверяет действия, поддерживая единообразный и справедливый игровой процесс для всех участников. Централизованный контроль состояния также помогает противодействовать читерству.

Клиенты и серверы могут взаимодействовать через интернет или локальную сеть (LAN). В офлайн-играх по LAN несколько находящихся рядом устройств соединяются без доступа к интернету. Такая схема обычно обеспечивает минимальную задержку, высокую защищенность и надежное соединение благодаря физической близости устройств. Она подходит для LAN-вечеринок, киберспортивных турниров и мест с нестабильным интернетом.

В клиент-серверной топологии применяются серверы двух типов:

Выделенный игровой сервер

Выделенный игровой сервер - отдельный узел, который только обрабатывает данные и не участвует в игре как игрок. Он способен обеспечить максимальную производительность, выполняя все основные симуляции и обрабатывая взаимодействия участников сетевой игры.



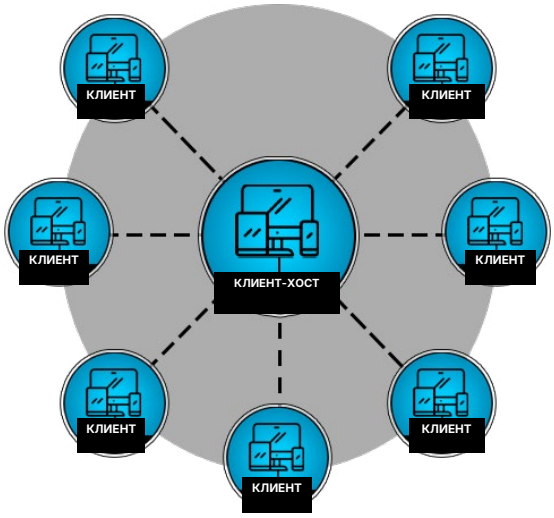
Выделенные серверы особенно важны для игр, где критична защита от читерства. Однако такая схема может увеличить задержку обмена: каждое изменение состояния игрока должен обработать сервер, прежде чем передать его остальным клиентам.

Выделенные серверы хорошо подходят для чувствительных к производительности соревновательных игр, например шутеров от первого лица. Они помогают поддерживать равные условия и ограничивать недобросовестное поведение.

Выделенный игровой сервер обрабатывает все основные игровые симуляции.

Listen-сервер на клиенте

Listen-сервер на клиенте одновременно выступает сервером и клиентом, поэтому хост может участвовать в игре. Это снижает расходы, но дает хосту преимущество по задержке, поскольку его пакетам не приходится проходить через сеть.



Производительность такого сервера часто ниже: один компьютер одновременно выполняет серверную часть и формирует изображение для игрока-хоста. Кроме того, хост обслуживает соединения через домашний интернет, который может уступать выделенному серверу в удаленном центре обработки данных. Домашние провайдеры обычно оптимизируют скорость загрузки сильнее, чем скорость отправки данных.

Listen-сервер на клиенте одновременно работает как сервер и как клиент.

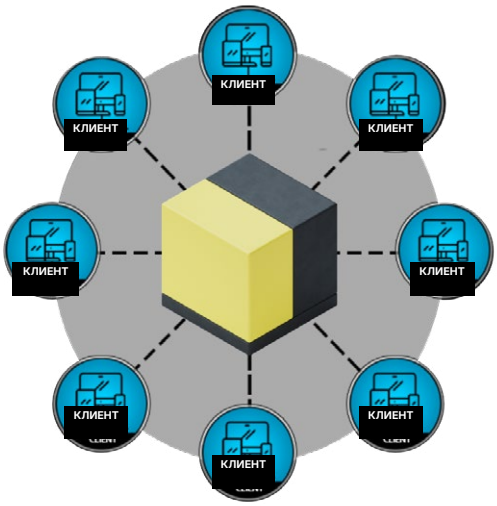
Распределенные полномочия

Модель Distributed Authority децентрализует контроль и управление состоянием игры между участвующими клиентами. Каждый клиент владеет частью игровых объектов, отслеживает их состояние и может самостоятельно создавать эти объекты и управлять ими.

Центральный облегченный сервис отслеживает изменения состояния объектов и маршрутизирует сетевой трафик, но не выполняет симуляцию самой игры.

У этой топологии есть несколько преимуществ: более низкая стоимость и меньшая задержка ввода. Центральному серверу не приходится обрабатывать все игровые действия, а число сетевых циклов приема-передачи сокращается, поскольку каждый клиент имеет полномочия над собственными объектами. Например, движение, атаки и другие команды можно обработать локально, не ожидая разрешения сервера. Игрок быстрее видит результат, и управление ощущается отзывчивее. Однако отсутствие единого авторитетного сервера, проверяющего все действия, повышает уязвимость к читерству.

Distributed Authority хуже подходит для строго детерминированных или высококонкурентных игр, но хорошо работает в проектах, где точность взаимодействия не столь критична.



Модель Distributed Authority децентрализует контроль и управление игрой.

Подробнее о новом beta-пакете Unity Distributed Authority для Netcode for GameObjects можно узнать из доклада Unite 2024.

Локальная многопользовательская игра на одном экране

В локальной многопользовательской игре один экземпляр клиента используется двумя или несколькими людьми, которые находятся в одном месте и играют на общем экране. Сетевое соединение не требуется, поэтому участники могут взаимодействовать напрямую. Такая схема популярна в играх для компаний и кооперативных режимах: друзьям и семье легко играть вместе.

Одноранговая сеть (P2P)

Каждое устройство одновременно работает как клиент и сервер, обеспечивая прямое соединение между игроками. Эта схема напоминает Distributed Authority, но полностью обходится без облегченного центрального сервиса. Отказ от централизованных серверов снижает расходы и сложность, однако затрудняет обеспечение равных условий и стабильной задержки, поскольку единый центр управления состоянием и безопасностью отсутствует.

Что такое авторитетный сервер?

Авторитетным называют сервер, который централизованно управляет состоянием и логикой игры. Именно он принимает окончательные решения в сетевой игре.

Авторитетный сервер не распределяет полномочия между устройствами игроков, а сам выполняет полную симуляцию и определяет происходящее в игре. Клиенты лишь отправляют ему команды, после чего сервер обновляет игру и возвращает актуальное состояние.

Сервер также обеспечивает соблюдение правил и разрешает конфликты. Авторитетный сервер - один из самых простых способов реализовать сетевую игровую логику и один из наиболее устойчивых к эксплуатации читерами, поэтому все игроки получают единообразный опыт.

Сетевой стек

Стек протоколов, или сетевой стек, - это программное обеспечение, реализующее различные протоколы связи для передачи данных по сети. Он устроен подобно слоеному пирогу:

Сетевой стек (источник: Wikipedia)

Каждый уровень взаимодействует только с соседними уровнями сверху и снизу. Это обеспечивает модульность и упрощает управление сетью.

Прикладной уровень: верхняя часть стека, где выполняется основная работа разработчика Unity. Пакеты Netcode for GameObjects и Netcode for Entities скрывают сложность низкоуровневого сетевого взаимодействия, позволяя сосредоточиться на реализации многопользовательских возможностей.

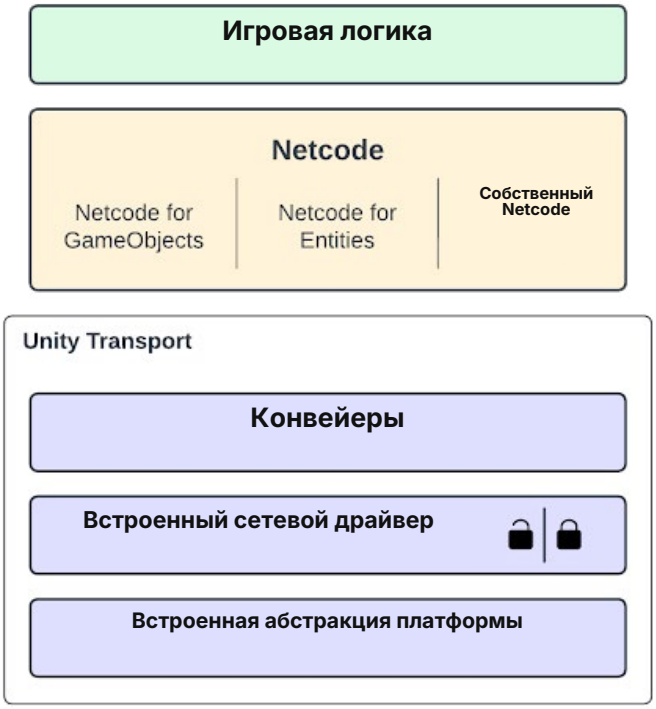
Транспортный уровень: отвечает за надежную передачу данных, обнаружение и исправление ошибок, управление потоком и сквозной обмен между устройствами сети. Он обеспечивает сегментацию и повторную сборку пакетов, восстановление после ошибок и целостность данных.

Сетевой уровень: отвечает за маршрутизацию пакетов между устройствами в разных сетях. Для обмена он использует сетевую инфраструктуру и протоколы, например IP (Internet Protocol - интернет-протокол).

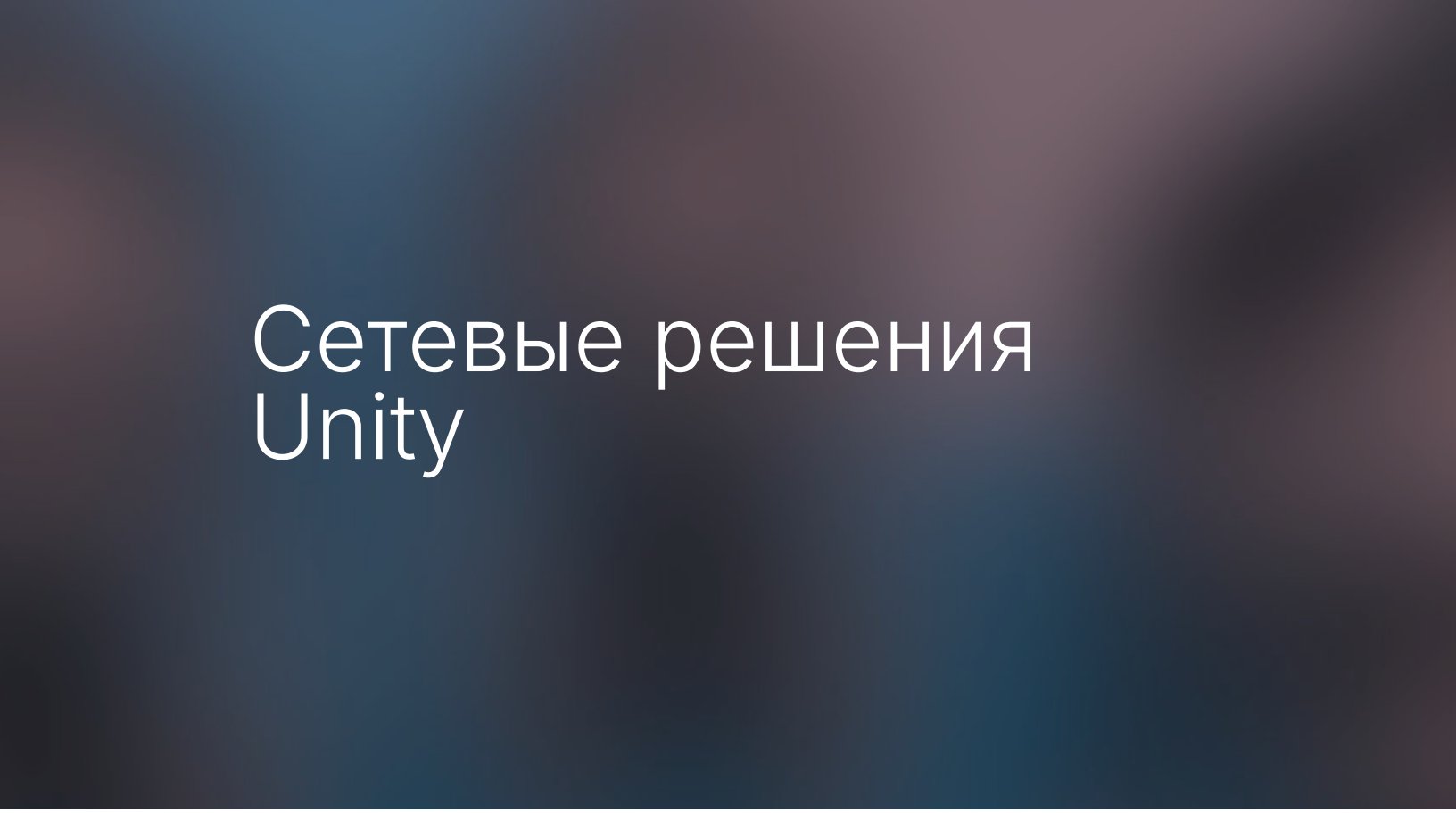
Канальный и физический уровни: непосредственно обеспечивают передачу пакетов по физической среде, например Ethernet или Wi-Fi. Обычно работу этих уровней выполняют операционная система и сетевое оборудование.

Разработчик Unity в основном работает с высокоуровневым прикладным уровнем, реализуя многопользовательские функции: синхронизацию GameObject, управление состоянием игры и обработку взаимодействий игроков.

Обычно нижние уровни не требуют внимания, если у приложения нет особых требований. Поэтому сетевой стек можно представить в упрощенном виде:



Уровни разработки сетевого кода



Сетевые решения Unity

Unity предлагает комплексные решения для многопользовательских игр разных жанров и масштабов. Важно правильно выбрать систему сетевого кода, поскольку требования к обмену данными зависят от типа игры.

Учитывайте жанр и масштаб игры, уровень соревновательности и необходимую степень контроля над сетевым уровнем. Для казуальных проектов часто важнее простота и экономичность, а соревновательным играм требуется точное и надежное управление сетью, обеспечивающее равные условия и отзывчивость.

Создаете ли вы казуальную кооперативную игру или соревновательный экшен, Unity предлагает производительные пакеты сетевого кода и дополнительные сервисы для решения ваших задач.

Netcode for GameObjects

Для казуальных кооперативных многопользовательских игр Unity рекомендует пакет Netcode for GameObjects (NGO). Это высокоуровневое решение скрывает детали сетевой логики и тем самым упрощает разработку и управление состоянием игры у всех участников.

Если вы только начинаете заниматься многопользовательской разработкой, NGO станет отличной отправной точкой.

Netcode for GameObjects включает несколько основных возможностей:

- NetworkObject: представляет любой игровой объект, который требуется синхронизировать по сети. Компонент управляет созданием и удалением объекта, а также владением им.

- NetworkBehaviour: специализированный MonoBehaviour с сетевыми возможностями. Он предоставляет встроенные обратные вызовы для сетевых событий и позволяет писать как серверный, так и клиентский код.

- Удаленные вызовы процедур (RPC): отправляют сообщения и вызывают методы удаленных экземпляров GameObject с помощью Server RPC и Client RPC.

- NetworkVariable: класс для синхронизации состояния по сети.

- NetworkManager: центральный компонент, который управляет сетевым состоянием игры, включая подключение и отключение клиентов, а также управление сценами.

Эти компоненты работают на прикладном уровне и помогают реализовать многопользовательские возможности. Netcode for GameObjects сочетает простоту с широкими возможностями и предоставляет инструменты для создания надежных, масштабируемых многопользовательских игр.

Netcode for Entities

Netcode for Entities построен на основе Data-Oriented Technology Stack (DOTS) и Entity Component System (ECS) Unity и предназначен для игрового процесса под управлением авторитетного сервера. Он поддерживает прогнозирование на стороне клиента, интерполяцию, компенсацию задержки и другие расширенные возможности.

В такой архитектуре централизованный сервер выполняет все игровые симуляции и контролирует результаты, ограничивая возможности для читерства. Клиенты отправляют серверу ввод, а он обрабатывает данные и возвращает обновленное состояние игры, сводя к минимуму возможность манипуляций.

Netcode for Entities использует прогнозирование на стороне клиента для компенсации задержки, обеспечивая более плавный игровой процесс с почти незаметным лагом. По сравнению с Netcode for GameObjects это решение лучше масштабируется и эффективнее использует пропускную способность.

Если вы опытный разработчик многопользовательских игр и проект требует высокой производительности и детерминизма, DOTS и ECS могут стать подходящей основой.

Подробнее об ориентированном на данные проектировании в Unity читайте в нашей электронной книге о DOTS и в подборке ресурсов DOTS.

Демонстрационный проект Unity ECS Network Racing создан с помощью Netcode for Entities.

Выбор решения Netcode зависит от требований проекта и опыта команды. Приведенная ниже таблица поможет принять первоначальное решение.

Характеристика	Netcode for GameObjects	Netcode for Entities
Целевая аудитория	Начинающие и разработчики среднего уровня	Опытные разработчики
Архитектура	Объектно-ориентированная, на основе MonoBehaviour	Ориентированная на данные: ECS и DOTS
Производительность	Для игр небольшого масштаба	Оптимизировано для высокой производительности и масштабируемости
Масштабируемость	Ограниченная; для малого числа игроков	Высокая; для крупных игр
Сетевые возможности	NetworkVariables, RPC, NetworkTransform, ограниченное прогнозирование клиента, интерполяция, UnityTransport	Полное прогнозирование клиента, интерполяция, компенсация задержки, оптимизированный UnityTransport
Интеграция Unity Services	Полная поддержка Lobby, Relay и других сервисов	Полная поддержка Lobby, Relay и других сервисов
Примеры проектов	Boss Room, Bite Size Samples	Megacity Metro, ECS Racing, Netcode Samples

Еще один полезный ресурс - Unity Multiplayer Center, отправная точка для создания многопользовательской игры. Он рекомендует пакеты Unity с учетом потребностей проекта и предоставляет доступ к примерам и учебным материалам. Инструкции по установке и настройке пакета приведены в документации Multiplayer Center.

Unity Transport

Unity Transport Package - независимая от конкретной реализации сетевого кода библиотека, предоставляющая низкоуровневый сетевой слой с упором на производительность и надежность. Эта современная, защищенная и переносимая транспортная библиотека расширяет возможности обычного UDP:

- Надежность: обеспечивает гарантированную связь поверх UDP, доставляя критически важные сообщения без накладных расходов TCP.

- Безопасность: использует шифрование и аутентификацию для защиты данных от несанкционированного доступа.

- Производительность: оптимизирована для низкой задержки и высокой пропускной способности.

- Кроссплатформенность: рассчитана на бесперебойную работу на разных платформах и устройствах.

По умолчанию и Netcode for GameObjects, и Netcode for Entities используют Unity Transport. Разработчики, которым нужен точный контроль над сетью, могут применять Unity Transport как самостоятельную библиотеку и создавать поверх нее собственный сетевой код для конкретных задач игры.

В Unity Transport появилась поддержка WebGL, расширяющая возможности кроссплатформенных многопользовательских веб-игр.

Сторонние сетевые решения

Мы рекомендуем начинать с Netcode for GameObjects, однако это не единственный вариант. Сообщество Unity предлагает множество решений для самых разных задач.

В Unity Asset Store доступны, в частности, следующие сторонние сетевые решения:

- Photon Unity Networking (PUN): комплексное масштабируемое решение для игр с большим числом одновременно подключенных пользователей.

- Mirror: простая и удобная бесплатная сетевая библиотека с открытым исходным кодом, изначально созданная на основе Unity UNet.

- DarkRift Networking 2: производительное и гибкое решение для разработчиков, которым требуется детальный контроль над сетевым взаимодействием.

- Forge Networking Remastered: решение с открытым исходным кодом, предоставляющее расширенные возможности настройки и контроля.

Настройка первого проекта Netcode

Если вы еще не работали с сетевыми решениями Unity, для создания базового проекта Netcode потребуется импортировать необходимые сетевые пакеты и настроить многопользовательские компоненты.

В этой главе мы добавим сетевое взаимодействие в демонстрационный проект с помощью Netcode for GameObjects. Напомним, что в Unity 6 новый многопользовательский проект можно настроить через Multiplayer Center, а дополнительные сервисы Unity интегрировать с помощью Multiplayer Widgets.

Прежде чем начать

Убедитесь, что у вас есть:

- Активная учетная запись Unity с действующей лицензией.

- Unity Hub

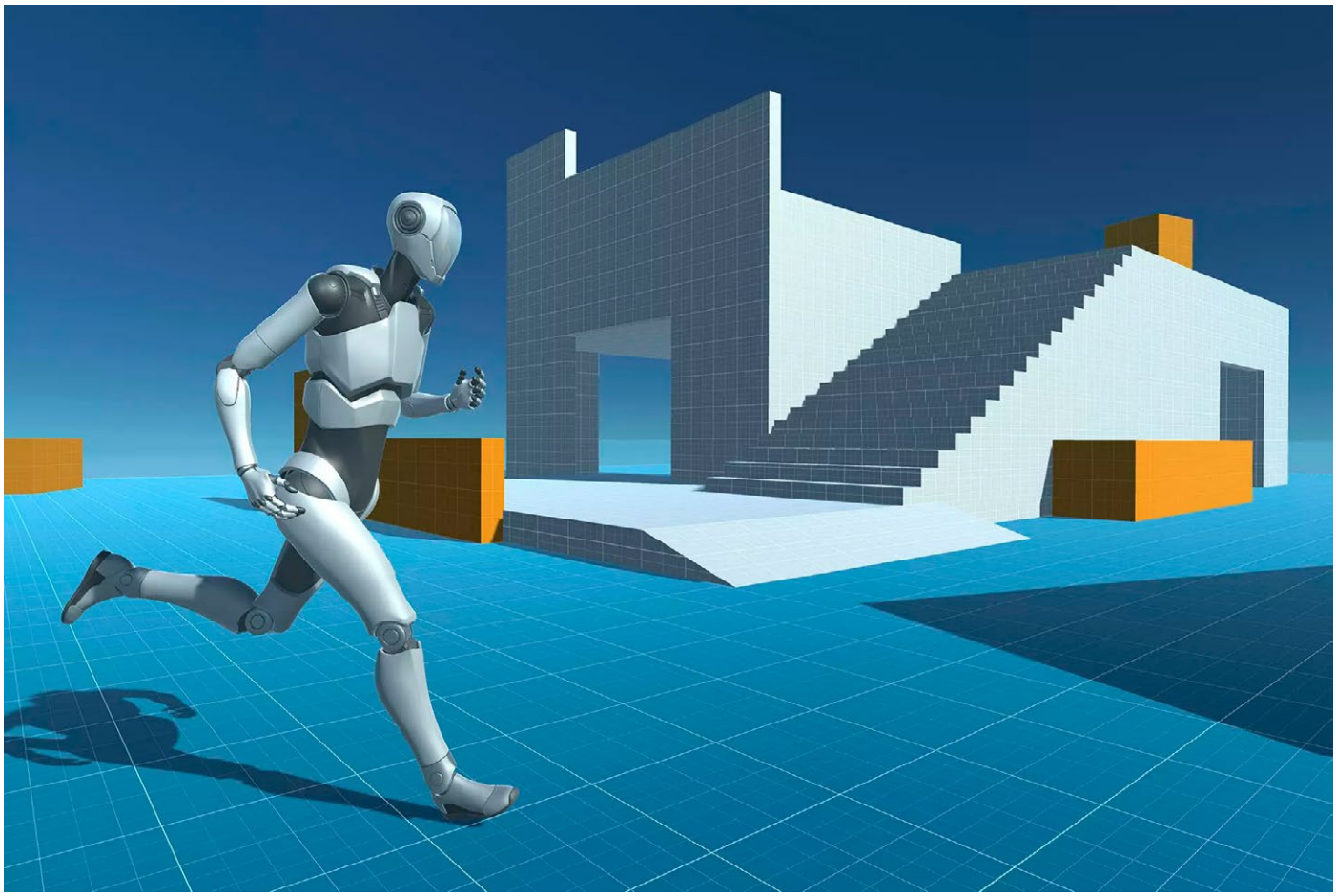
- Поддерживаемая версия Unity Editor. Для некоторых показанных возможностей требуется Unity 6 или новее; подробности см. в требованиях Netcode for GameObjects.

- Доступ к Unity Cloud Dashboard для подключения необходимых проекту сервисов Unity; настроить его можно через Unity Hub.

Настройка демонстрационного проекта

Удобнее всего знакомиться с инструментами сетевого кода на существующем проекте с однопользовательским управлением персонажем. В этом руководстве используется пакет Starter Assets - ThirdPerson из Unity Asset Store. Он демонстрирует простой трехмерный игровой процесс с гуманоидным персонажем на Universal Render Pipeline (URP).

Загрузите этот бесплатный ресурс из Unity Asset Store, затем импортируйте его с помощью Package Manager.



Пакет Starter Assets из Unity Asset Store

Демонстрационный проект включает небольшую тестовую сцену и настраиваемый контроллер от третьего лица. Наша цель - запустить несколько экземпляров приложения, чтобы разные клиенты могли взаимодействовать в общей среде.

Установка Netcode for GameObjects

В Package Manager (Window > Package Manager) установите фильтр Unity Registry, затем установите следующие пакеты:

- Netcode for GameObjects: базовая сетевая библиотека, которая добавляет многопользовательские возможности в существующий рабочий процесс GameObject/MonoBehaviour. Она упрощает разработку сетевых игр и служит отличной отправной точкой.

- Multiplayer Tools Window: дополнительный набор из пяти инструментов Unity 6, улучшающих рабочие процессы многопользовательской разработки:

    - Multiplayer Tools Window предоставляет в одном месте удобный доступ ко всем многопользовательским инструментам и их документации.

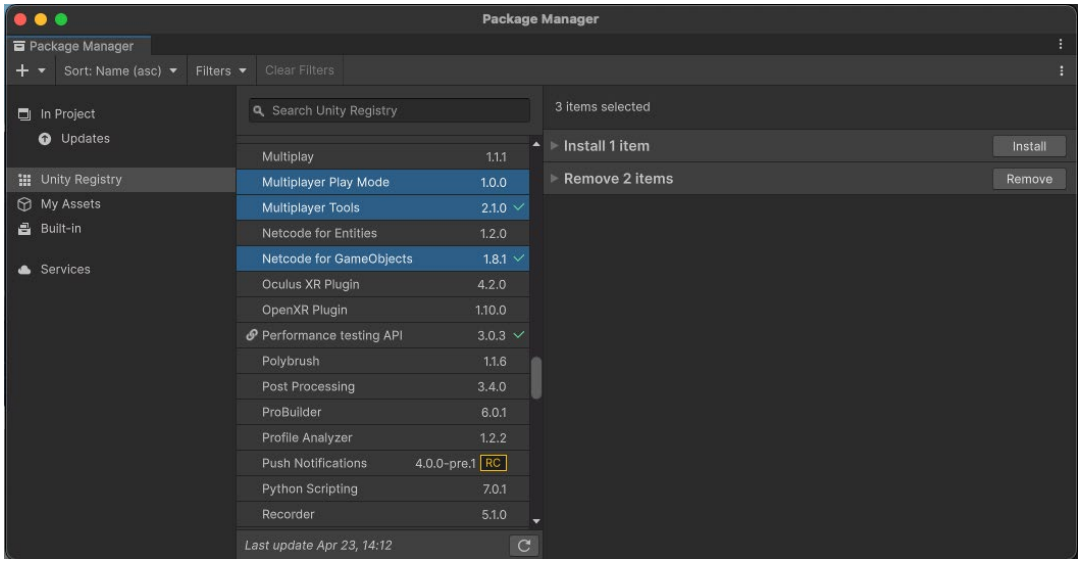
    - Network Simulator воспроизводит реальные сетевые условия - задержку и потерю пакетов, разрывы соединения - и помогает выявить проблемы до выпуска игры.

    - Runtime Network Stats Monitor (RNSM) отображает сетевую статистику в реальном времени и позволяет настраивать экранный мониторинг производительности сети.

    - Network Scene Visualization упрощает отладку, наглядно отображая сетевую активность и владение объектами в окне Scene.

    - Hierarchy Network Debug добавляет в правую часть окна Hierarchy слой, который отмечает сетевые объекты небольшим значком сетевого куба.

- Multiplayer Play Mode: пакет Unity 6 для тестирования многопользовательских возможностей без выхода из Unity Editor. Он позволяет имитировать до четырех игроков - Main Editor Player и трех Virtual Players - и ускоряет проверку игрового процесса.



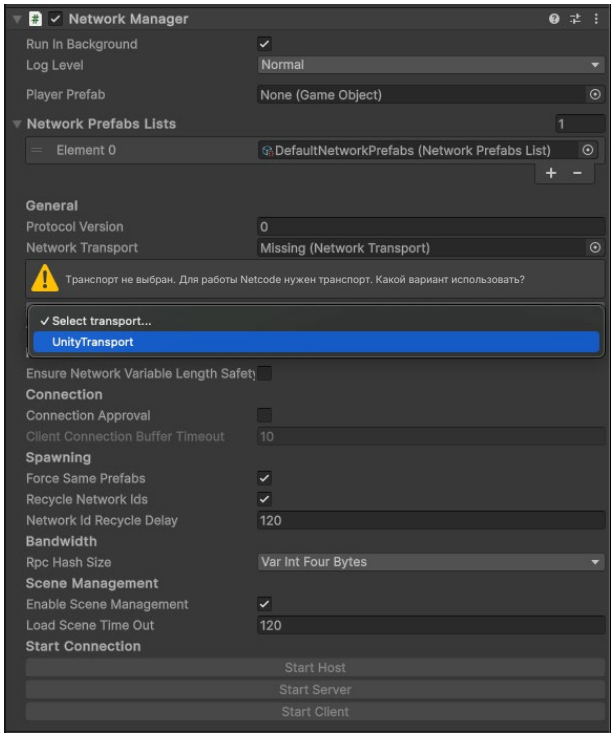
Установите Netcode for GameObjects и его вспомогательные пакеты.

Добавление NetworkManager

Для поддержки сетевой многопользовательской игры каждому проекту нужен компонент NetworkManager. Он управляет сетевым состоянием проекта, соединениями и параметрами сети.

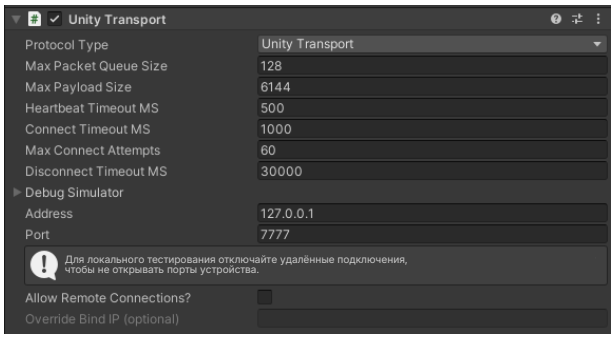
Чтобы добавить NetworkManager в сцену, создайте в окне Hierarchy новый GameObject и добавьте компонент NetworkManager (Netcode > NetworkManager).

В компоненте NetworkManager настройте сетевой транспорт, выбрав Unity Transport.



Выберите транспортный слой в NetworkManager.

К GameObject будет добавлен компонент UnityTransport. Транспортный слой отвечает за низкоуровневые сетевые задачи, включая управление соединениями, передачу данных и шифрование пакетов.



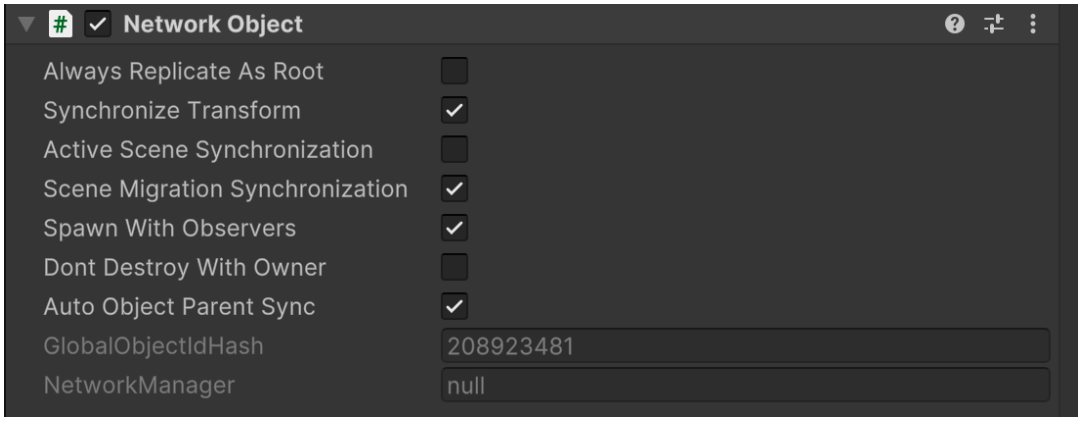
Компонент UnityTransport

Пока изменять эти параметры не требуется, но с помощью компонента можно имитировать сетевые условия - задержку, потерю пакетов и jitter - при тестировании и отладке в Unity Editor.

Сохраните сцену, откройте File > Build Settings и убедитесь, что текущая сцена добавлена в список Scenes in Build. Тогда новый NetworkManager войдет в сборку игры.

NetworkObjects

NetworkObject - обязательный компонент любого GameObject, который должен быть доступен по сети или синхронизироваться между клиентами многопользовательской игры. После добавления NetworkObject объект становится сетевым: его состояние и поведение можно передавать и обновлять по сети.



Компонент NetworkObject и его уникальный идентификатор

У каждого NetworkObject есть несколько идентификаторов:

- GlobalObjectIdHash идентифицирует ресурс префаба в проекте.

- NetworkObjectId - уникальный идентификатор, различающий экземпляры одного и того же префаба.

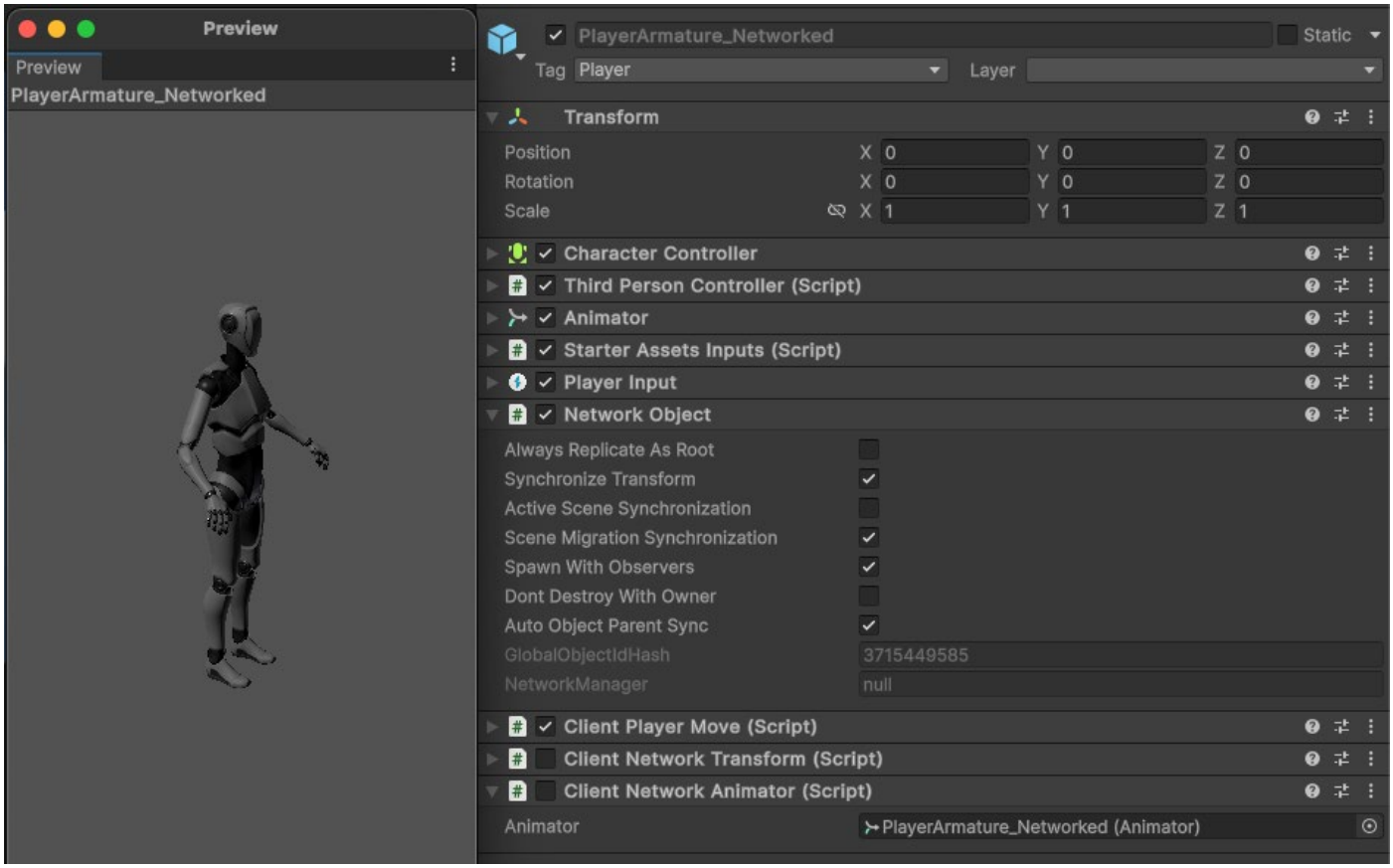
- OwnerClientId указывает клиента, которому принадлежит объект (см. раздел о полномочиях ниже).

Эти идентификаторы позволяют NetworkManager отслеживать объект и поддерживать согласованность его состояния у всех подключенных клиентов. Объекты NetworkObject можно динамически создавать (spawn) и уничтожать во время игры.

При создании NetworkObject появляется у всех подключенных клиентов. У каждого такого объекта есть владелец - обычно клиент, управляющий его поведением и состоянием.

Объекты Player NetworkObject

У каждого игрока может быть собственный префаб Player NetworkObject. Это особый вид NetworkObject, который обычно содержит контроллер персонажа и его визуальное представление в игре.



Player NetworkObject в демонстрационном проекте

Объекты Player NetworkObject часто хранят и синхронизируют данные конкретного игрока: имя, счет, инвентарь и другие сведения. Синхронизация по сети обеспечивает всем подключенным игрокам согласованное представление состояния игры.

При подключении клиента NetworkManager создает принадлежащий соответствующему игроку Player NetworkObject. Игрок получает полномочия над своим PlayerObject и может управлять его поведением и состоянием.

Чтобы настроить Player NetworkObject, сначала создайте в проекте обычный префаб GameObject. Он станет шаблоном PlayerObject и будет содержать компоненты и скрипты, определяющие поведение и внешний вид игрока.

Затем добавьте необходимые компоненты Netcode. Например:

- NetworkObject: каждому сетевому объекту требуется компонент NetworkObject. Он содержит свойства и события, связанные с созданием, удалением и владением объектом.

- NetworkBehaviour: эти скрипты добавляют сетевую логику к базовому классу MonoBehaviour. Они содержат NetworkVariable, удалённые вызовы процедур (RPC) и сетевые обратные вызовы.

- NetworkAnimator: этот компонент синхронизирует состояния и параметры анимации между клиентами.

- NetworkTransform: этот компонент обеспечивает репликацию позиции, поворота и масштаба игрока с сервера на все подключённые клиенты в реальном времени.

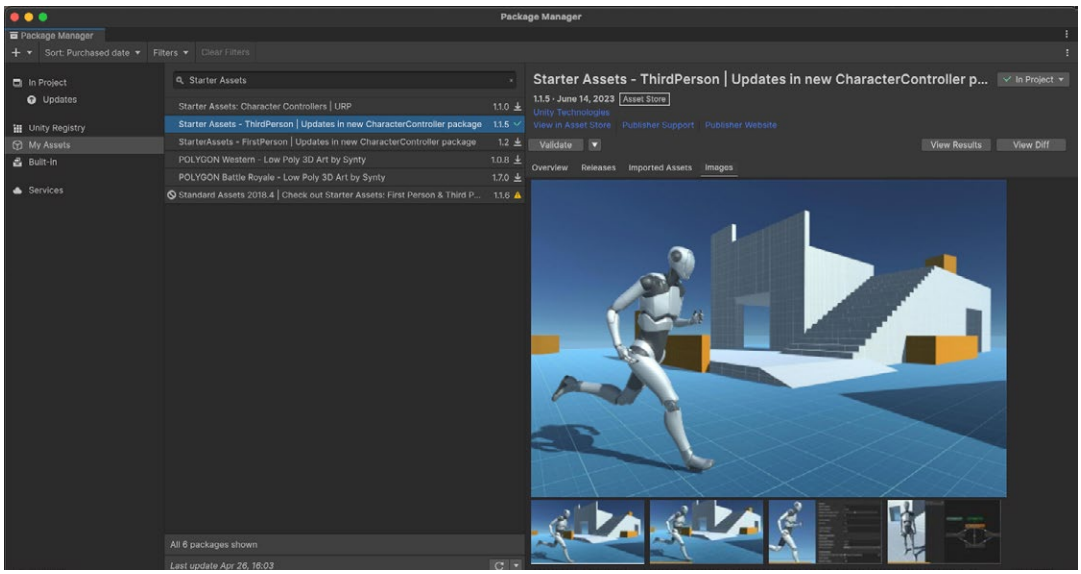
NetworkObject игрока часто отвечает за обработку его ввода. Когда игрок выполняет действие - например, перемещается или взаимодействует с игровым миром, - ввод обрабатывается и при необходимости передаётся другим подключённым игрокам.

Логика игрока сочетает MonoBehaviour для обычной игровой механики и NetworkBehaviour для управления сетевым состоянием. Несетевые компоненты, такие как контроллер персонажа и Animator, работают обычным образом в локальном экземпляре каждого игрока.

Локальная работа этих компонентов не только повышает производительность, но и сокращает сетевой трафик, что особенно важно при ограниченной пропускной способности соединений между клиентами.

Создание NetworkObject игрока

Откройте сцену Playground из примера проекта.

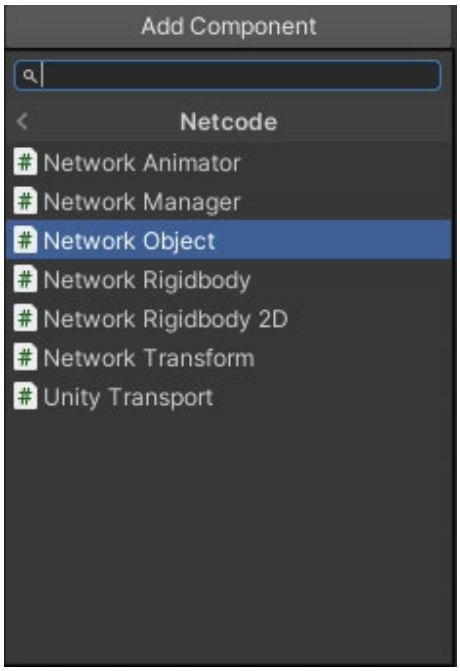


Пакет Starter Assets содержит сцену Playground.

В Hierarchy находится объект PlayerArmature, который управляет игровым персонажем. Чтобы превратить его в NetworkObject игрока, перетащите объект из Hierarchy и создайте новый Original Prefab либо измените копию существующего префаба проекта.

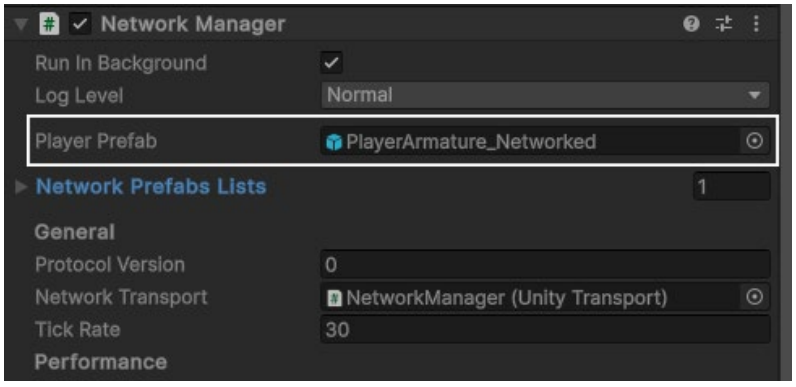
Найдите GameObject PlayerArmature в окне Hierarchy. Удалите его вместе с дочерними объектами, чтобы в сцене осталось только игровое окружение.

Затем откройте префаб в Inspector и добавьте компонент NetworkObject. Благодаря ему объект распознаётся и управляется по сети.



Добавьте компонент NetworkObject в префаб.

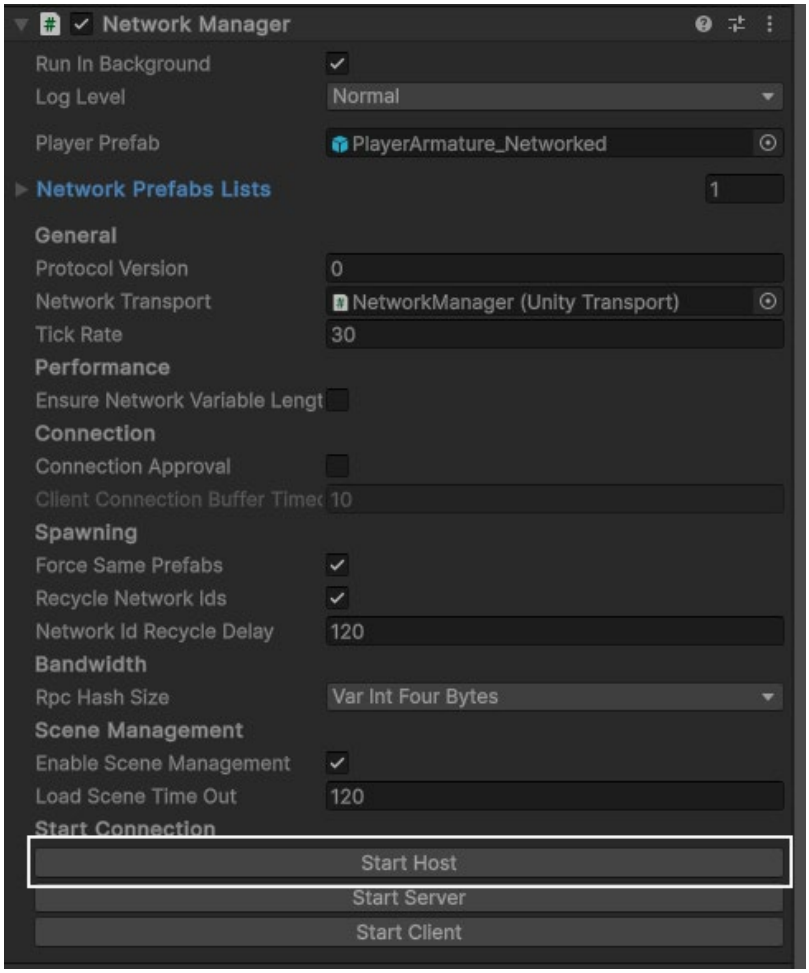
Укажите NetworkObject игрока в поле Player Prefab компонента NetworkManager.



Зарегистрируйте NetworkObject игрока в NetworkManager.

В Play mode отображается только игровое окружение. NetworkObject игрока появится лишь после подключения клиента.

Выберите NetworkManager, который теперь находится в разделе DontDestroyOnLoad окна Hierarchy.



Запустите хост через NetworkManager.

Нажмите Start Host. После этого будет создан сетевой объект игрока.

В Hierarchy появится объект PlayerArmature_Network. Игрой снова можно управлять, хотя цель камеры отключена. Проверьте перемещение игрока клавишами WASD.

После выхода из Play mode персонаж исчезнет. Теперь NetworkManager создаёт этого персонажа во время выполнения и управляет им.

Сетевую составляющую игры невозможно оценить, пока не подключено несколько клиентов. Чтобы увидеть работу многопользовательской сцены, протестируем проект с несколькими клиентами.

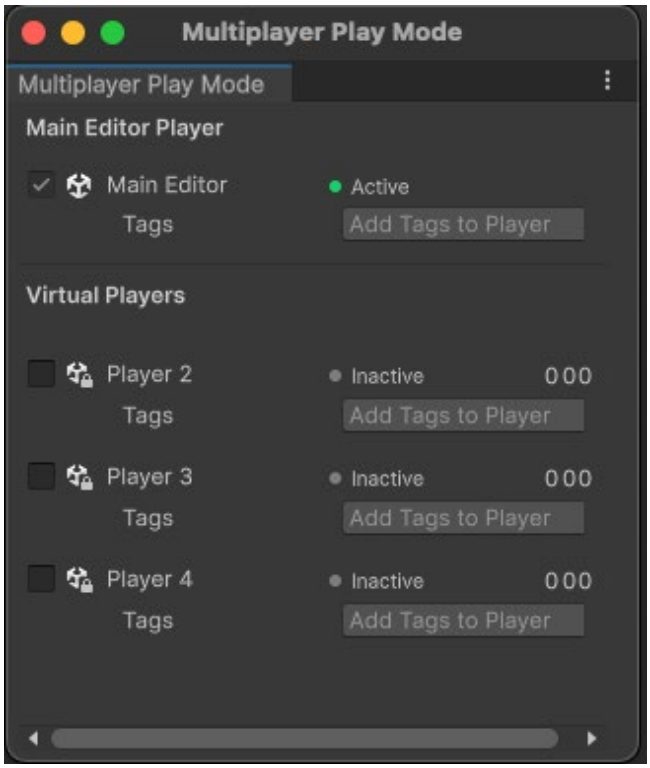
Multiplayer Play Mode

Для тестирования многопользовательской игры приложение должно работать в нескольких отдельных процессах. Раньше для этого приходилось создавать отдельную сборку игры и запускать её параллельно с Unity Editor.

Этот способ по-прежнему доступен, однако в Unity 6 есть Multiplayer Play Mode (MPPM). Он позволяет одновременно открыть несколько экземпляров Unity Editor и воспроизвести многопользовательскую среду, тем самым упрощая тестирование.

Установите Multiplayer Play Mode через Package Manager. После этого для проверки каждой новой функции не придётся заново собирать приложение.

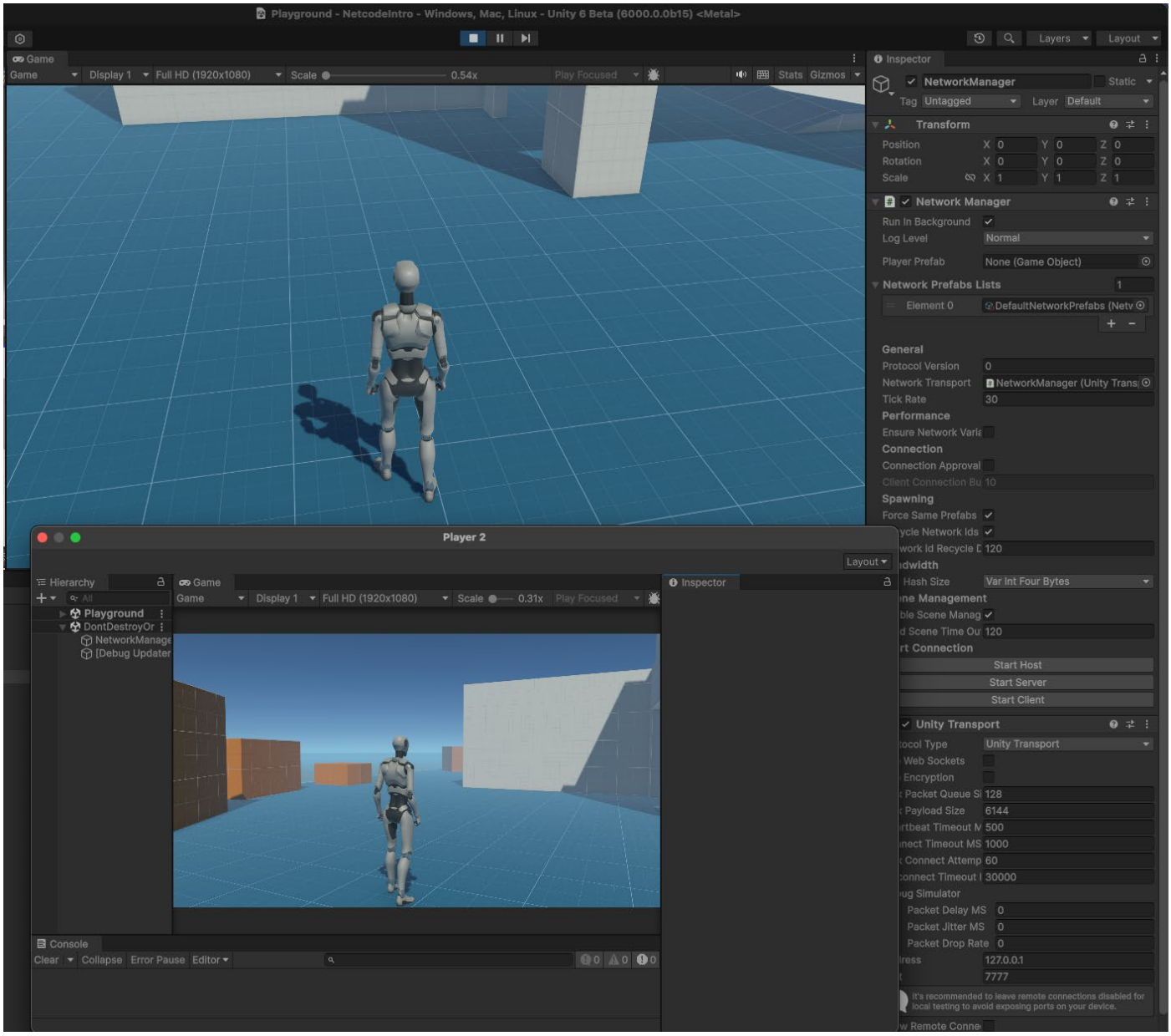
Откройте Multiplayer Play Mode: Window > Multiplayer Play Mode.



Окно Multiplayer Play Mode

Затем включите в списке на снимке выше хотя бы один дополнительный Virtual Player. Так вы сможете протестировать как минимум один хост и один клиент. Помните: хост - это клиент, который одновременно выполняет роль сервера.

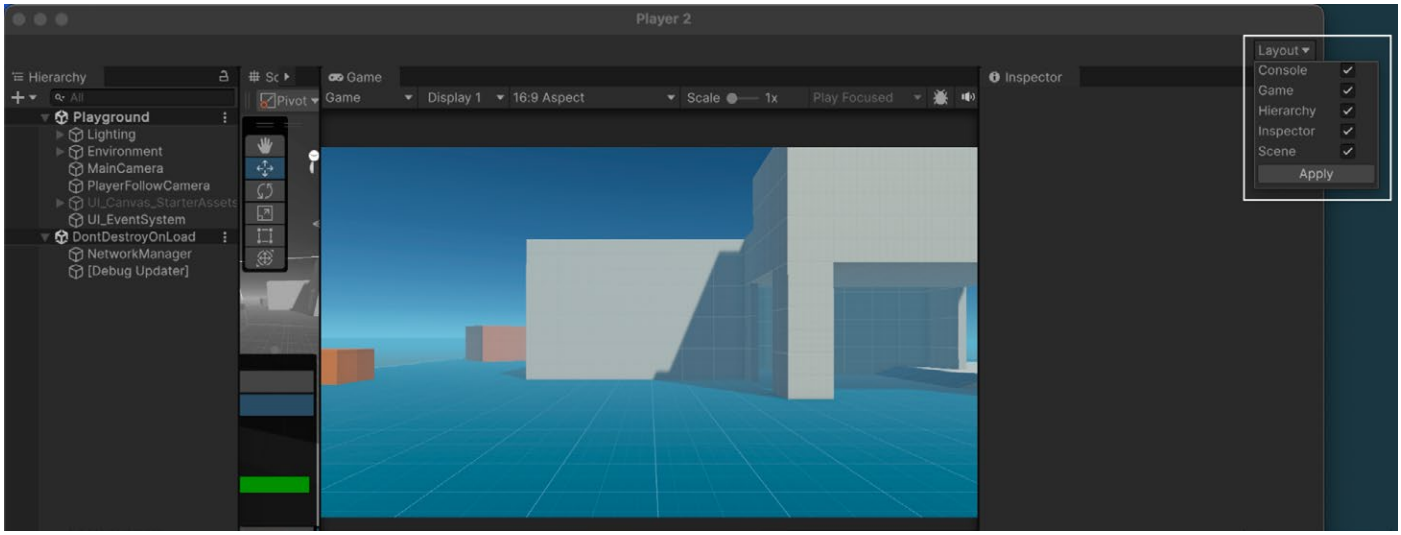
При переходе в Play mode второй сеанс приложения запускается в клонированном окне.



Multiplayer Play Mode клонирует Virtual Player.

Выберите NetworkManager в Hierarchy. В разделе Start Connection окна Inspector нажмите Start Host. В окне Game появится объект PlayerArmature_Networked.

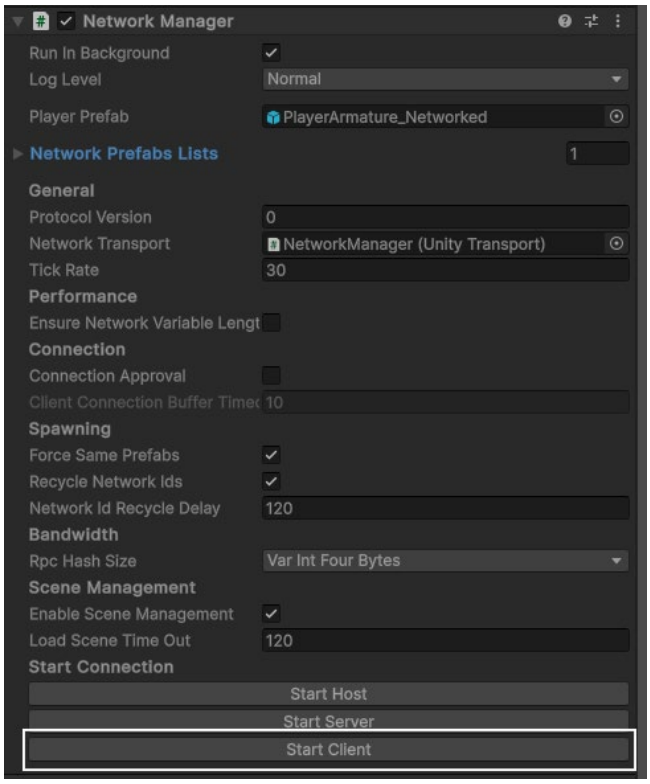
Во втором окне нажмите Layout и включите Inspector и Hierarchy, чтобы оно выглядело как ещё один сеанс Editor. Выберите нужные элементы интерфейса и нажмите Apply.



Включите нужные Layouts в клонированном окне.

В клонированном окне Editor найдите NetworkManager в разделе DontDestroyOnLoad окна Hierarchy.

В Inspector, в разделе Start Connection, нажмите Start Client.



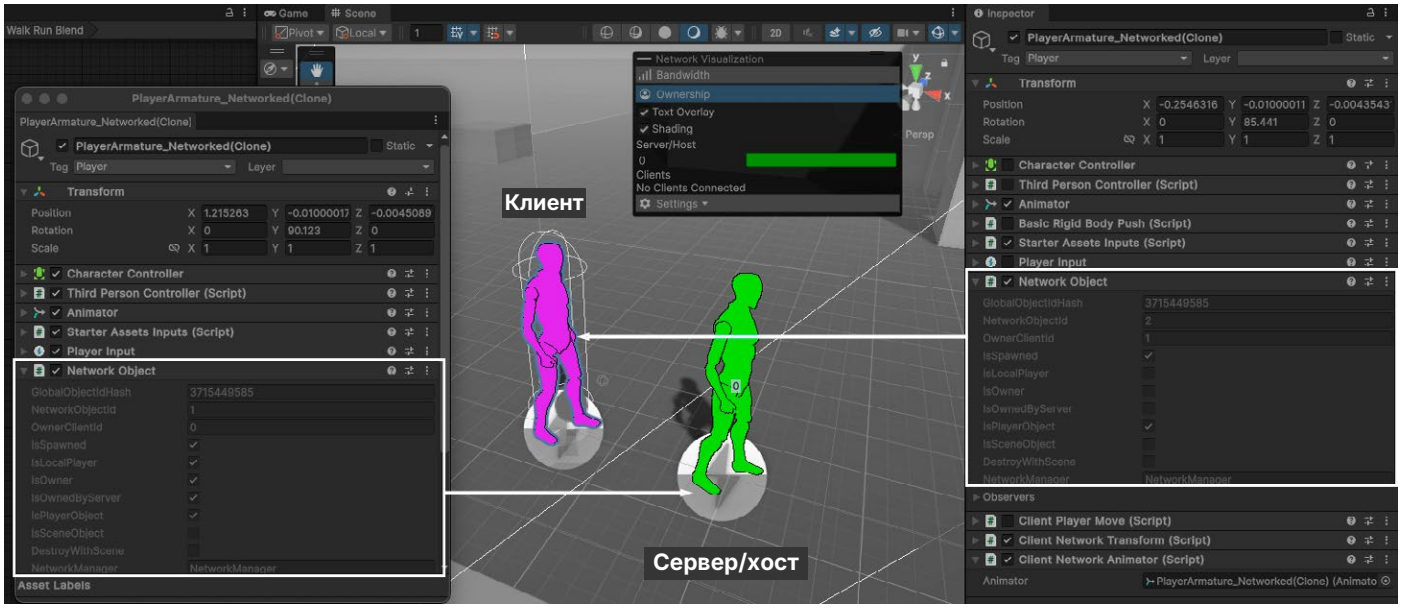
NetworkManager содержит кнопки для подключения клиента.

Теперь в Hierarchy отображаются два экземпляра PlayerArmature_Networked. В окне Scene они расположены друг на друге. Переместите один экземпляр с помощью клавиатуры или геймпада, чтобы разделить игроков.

Выберите экземпляр PlayerArmature_Networked в Hierarchy и изучите его компонент NetworkObject.

Во время выполнения каждый экземпляр определяется сочетанием GlobalObjectIdHash - идентификатора ресурса проекта - и NetworkObjectId - уникального индекса экземпляра. Ниже свойство OwnerClientId показывает, кто управляет экземпляром: хост или клиент.

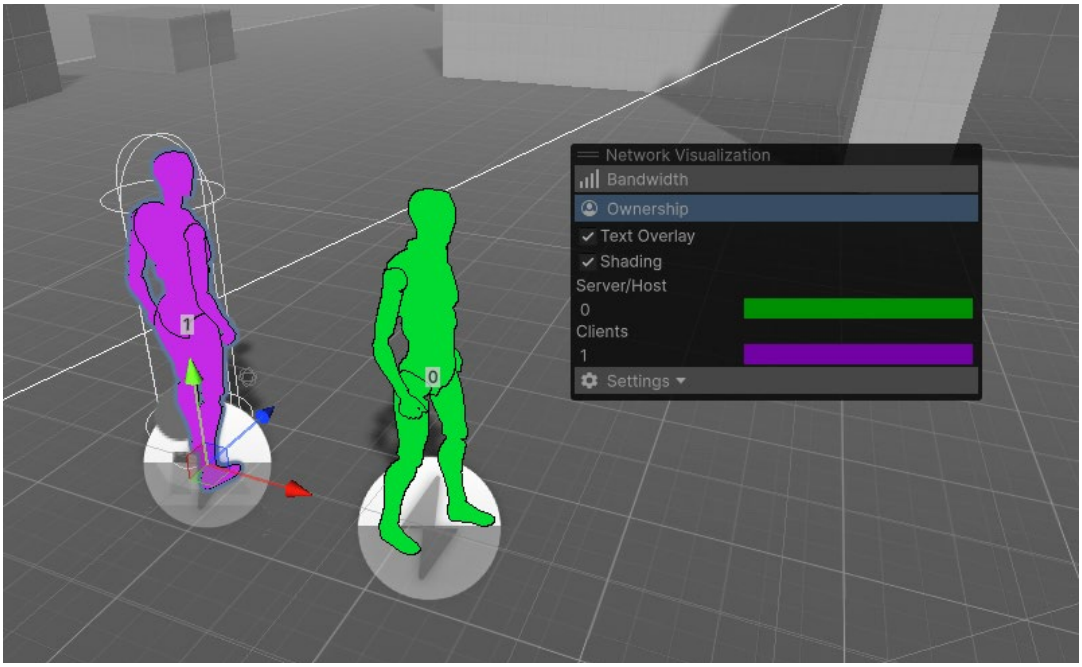
Переключайтесь между экземплярами и сравните флаги IsSpawned, IsLocalPlayer, IsOwner, IsOwnerByServer и другие.



Сравните настройки NetworkObject двух экземпляров.

Чтобы легче различать экземпляры, используйте панель Network Visualization. Этот удобный диагностический инструмент появляется в окне Scene после установки пакета Multiplayer Tools.

Экземпляры кодируются цветом по объёму трафика - Bandwidth - или по владению - Ownership, которое показывает, какой клиент имеет полномочия над NetworkObject игрока.



Network Visualization помогает отлаживать сетевые объекты.

NetworkManager создаёт отдельный экземпляр для каждого клиента, однако каждый из них независимо управляет одним и тем же персонажем. Перемещение с помощью WASD в окне Scene пока не синхронизируется между клиентом и хостом.

При первом подключении игроков NetworkManager синхронизирует их позиции в точке (0, 0, 0), но дальнейшие перемещения уже не синхронизируются. Сейчас поведением персонажа управляют несколько локальных компонентов:

- CharacterController позволяет игроку перемещаться и взаимодействовать с игровым окружением без сложных физических расчётов.

- Animator воспроизводит анимацию на основе машины состояний и управляет переходами и смешиванием состояний бега, прыжка и бездействия.

- PlayerInput управляет вводом отдельно для каждого игрока, сопряжением устройств и уведомлениями о событиях, предоставляя высокоуровневую оболочку над Unity Input System.

- StarterAssetsInputs преобразует этот ввод в значения перемещения, обзора, прыжка и спринта персонажа.

Это компоненты для одиночной игры. Чтобы они работали в многопользовательском приложении, необходимо добавить сетевую логику.

Создание собственных кнопок запуска сеанса

Чтобы удобнее запускать сетевые сеансы во время выполнения, добавьте экранные кнопки, которые повторяют функции кнопок NetworkManager в Inspector. Для этого подойдут Unity UI (UGUI) или UI Toolkit.

Создайте в выбранной системе интерфейса три кнопки: Client, Host и Server. Назначьте им соответствующие методы синглтона NetworkManager:

- `NetworkManager.Singleton.StartClient`
- `NetworkManager.Singleton.StartHost`
- `NetworkManager.Singleton.StartServer`

Эти методы позволяют запускать сетевой сеанс без кнопок в окне Inspector.

Добавление NetworkBehaviour

Для управления компонентами MonoBehaviour объекта PlayerArmature_Networked можно использовать NetworkBehaviour.

NetworkBehaviour - специализированный тип MonoBehaviour для сетевой логики. Он предоставляет средства синхронизации действий и состояний между игровыми клиентами.

NetworkBehaviour поддерживает те же события жизненного цикла, что и MonoBehaviour, а также предоставляет сетевые возможности:

Методы RPC: NetworkBehaviour может использовать удалённые вызовы процедур (RPC) для обмена данными по сети. Такие методы помечаются атрибутом [Rpc]. Для отправки RPC на сервер или клиент используются соответственно [Rpc(SendTo.Server)] и [Rpc(SendTo.Client)].

- NetworkVariable: специализированная переменная для синхронного управления состоянием по сети. Изменения NetworkVariable на сервере автоматически передаются всем клиентам.

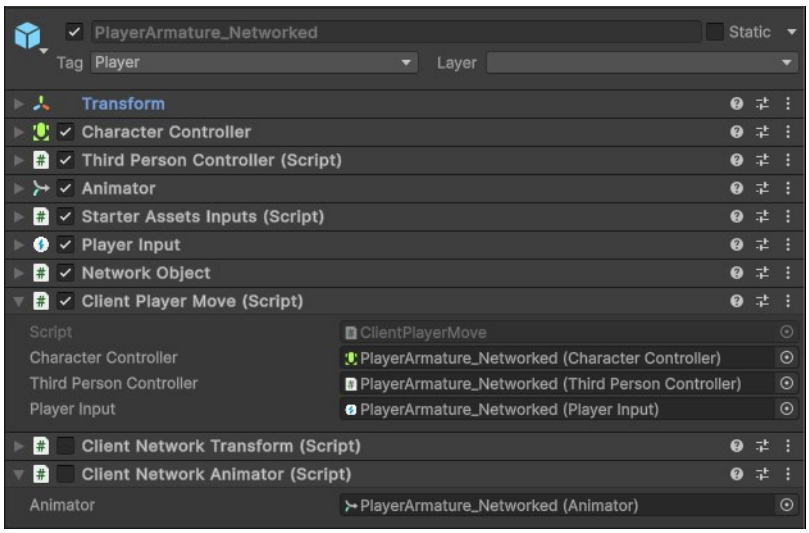
- OnNetworkSpawn и OnNetworkDespawn: методы жизненного цикла, вызываемые при создании и удалении NetworkBehaviour. OnNetworkSpawn выполняет инициализацию - подобно OnEnable или Start, но для сетевого поведения. OnNetworkDespawn очищает ресурсы перед удалением объекта из сети, аналогично OnDestroy или OnDisable.

- Владение: NetworkBehaviour позволяет назначить определённые объекты конкретным клиентам или серверу. Такая модель полномочий, при которой NetworkObject принадлежит клиенту либо серверу, гарантирует, что управлять объектом и взаимодействовать с ним смогут только назначенные игроки.

Для управления перемещением игрока реализуем NetworkBehaviour с именем ClientPlayerMove. Он гарантирует, что ввод хоста и клиента воздействует только на соответствующие им объекты игроков. Ниже приведён пример настройки:

```
using Unity.Netcode;
using StarterAssets;
using UnityEngine;
using UnityEngine.InputSystem;
namespace NetcodeDemo
{
   public class ClientPlayerMove: NetworkBehaviour
   {
       [SerializeField]
       CharacterController m_CharacterController;
       [SerializeField]
       ThirdPersonController m_ThirdPersonController;
       [SerializeField]
       PlayerInput m_PlayerInput;

       [SerializeField]
       Transform m_CameraFollow;
       private void Awake()

       {
           m_PlayerInput.enabled = false;
           m_ThirdPersonController.enabled = false;
           m_CharacterController.enabled = false;
       }

       public override void OnNetworkSpawn()
       {
           base.OnNetworkSpawn();
           enabled = IsClient; // Включить на клиенте.
           if (!IsOwner)
           {
               // Отключить, если объект не принадлежит клиенту
               enabled = false;
               m_PlayerInput.enabled = false;
               m_CharacterController.enabled = false;
               m_ThirdPersonController.enabled = false;
               return;
           }

           // Включить, если объект принадлежит клиенту
           m_PlayerInput.enabled = true;
           m_CharacterController.enabled = true;
           m_ThirdPersonController.enabled = true;

       }
   }
}
```

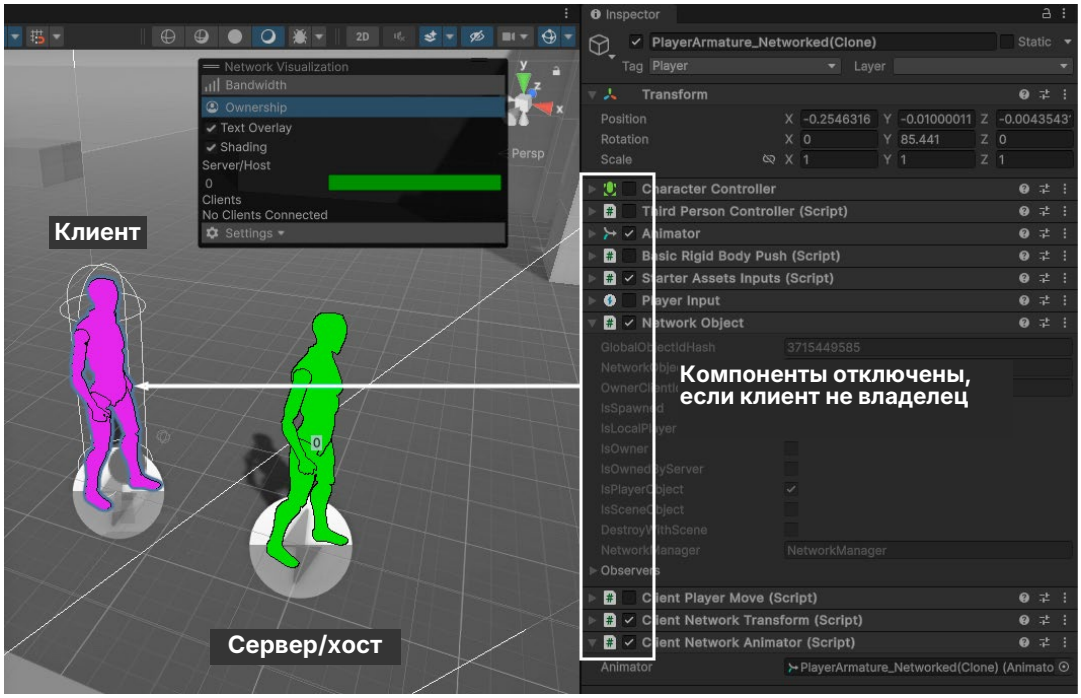
Добавьте этот компонент в префаб PlayerArmature_Networked, затем заполните нужные поля в Inspector.



Заполните поля ClientPlayerMove в Inspector.

Добавив скрипт в префаб, подключите сеансы хоста и клиента. При подключении клиентов к NetworkManager некоторые компоненты объекта игрока по умолчанию отключаются на основании свойства IsOwner: оно проверяет, принадлежит ли экземпляр локальному игроку.

В Hierarchy поочерёдно выбирайте два экземпляра PlayerArmature_Networked.



Некоторые компоненты отключаются, если экземпляр не принадлежит локальному клиенту.

Обратите внимание: у экземпляров игрока, которыми не владеет соответствующий клиент, теперь отключены некоторые компоненты, например PlayerInput. Хост управляет одним экземпляром игрока, а клиент - другим.

Теперь разными экземплярами игроков можно управлять независимо, однако их перемещение ещё не синхронизируется по сети. Чтобы хост и клиент видели одинаковое движение, добавим сетевые компоненты, например NetworkTransform.

Свойства полномочий и владения

По умолчанию NetworkObject принадлежат серверу, однако подключённый и одобренный клиент тоже может получить объект во владение с помощью метода SpawnWithOwnership. В Netcode for GameObjects сервер является авторитетной стороной: только он может создавать и удалять NetworkObject.

NetworkBehaviour предоставляет несколько свойств для быстрой проверки полномочий и владельца экземпляра:

- IsClient показывает, выполняется ли экземпляр на клиенте.

- IsServer показывает, выполняется ли экземпляр на сервере.

- IsHost показывает, выполняется ли экземпляр на хосте, который одновременно является сервером и клиентом.

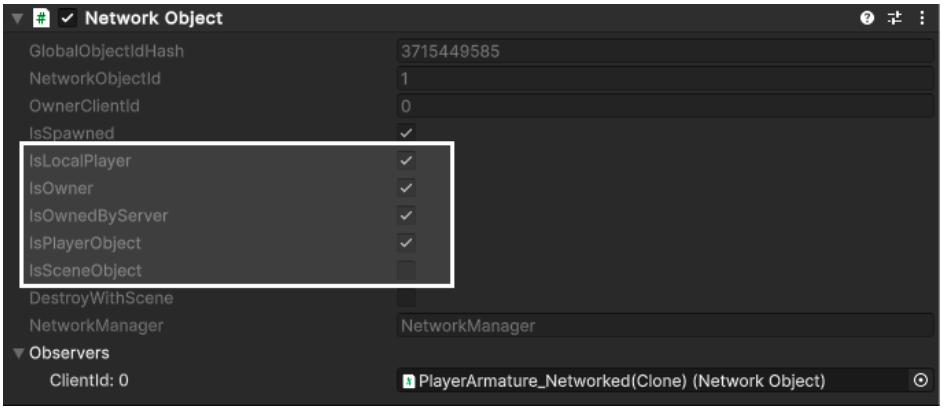
- IsLocalPlayer показывает, является ли связанный NetworkObject объектом локального игрока.

- IsOwner показывает, владеет ли локальный игрок этим объектом либо является ли объект локальным игроком.

- IsPlayerObject показывает, представляет ли GameObject сетевого игрока, которым обычно управляет определённый клиент.

- IsSceneObject показывает, является ли GameObject изначально частью сцены, а не создаётся динамически во время игры. Обычно сервер управляет объектами сцены, чтобы их состояние оставалось согласованным по сети.

При проверке NetworkObject во время выполнения отображаются некоторые из этих свойств.



Настройки NetworkObject

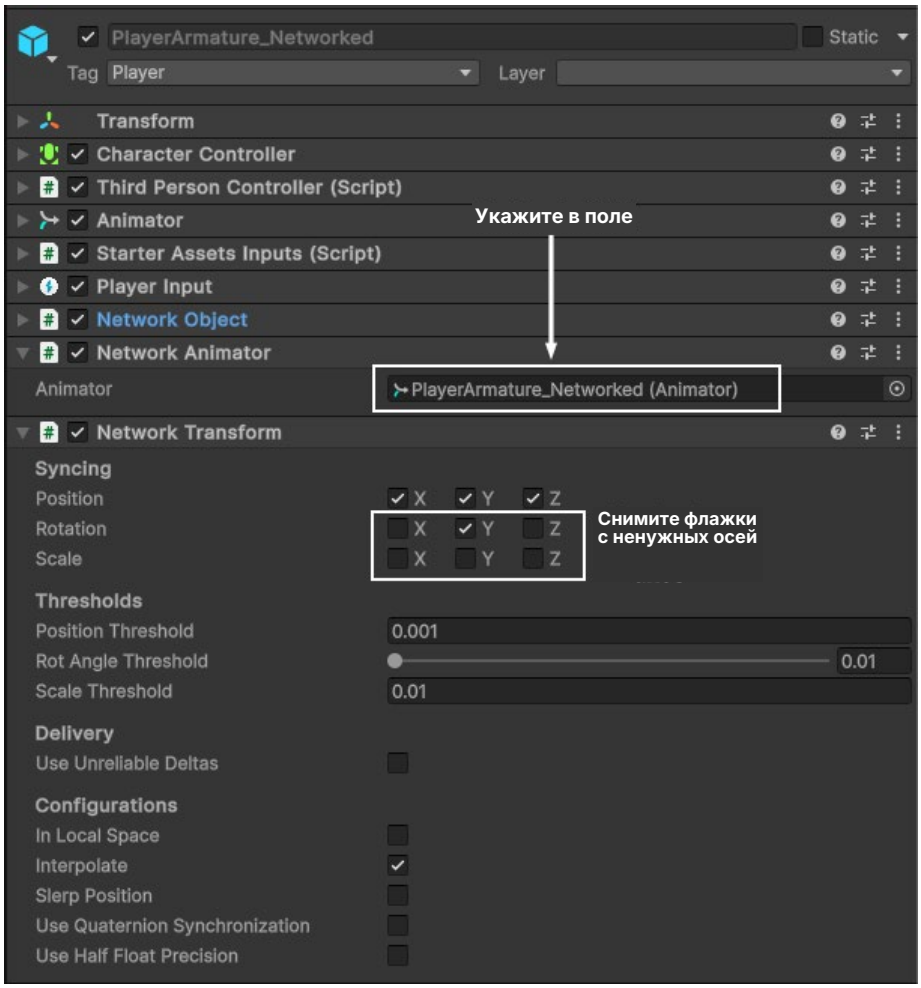
Синхронизация с помощью NetworkTransform и NetworkAnimator

NetworkBehaviour позволяет создать одинаковый экземпляр игрока на нескольких клиентах, но для синхронизации его перемещения нужны дополнительные компоненты.

Добавьте компонент NetworkTransform в префаб PlayerArmature_Networked. Отключите оси, которые не влияют на игровой процесс: в этом примере - синхронизацию масштаба, а также поворота по осям X и Z. Синхронизация расходует пропускную способность, поэтому не передавайте лишние данные.

В Multiplayer Play Mode активируйте окно хоста: теперь можно управлять игроком и наблюдать, как его движение синхронизируется с клиентом. Это первый шаг к сетевой игре.

Затем добавьте компонент NetworkAnimator в PlayerArmature_Networked и перетащите существующий компонент Animator в пустое поле.

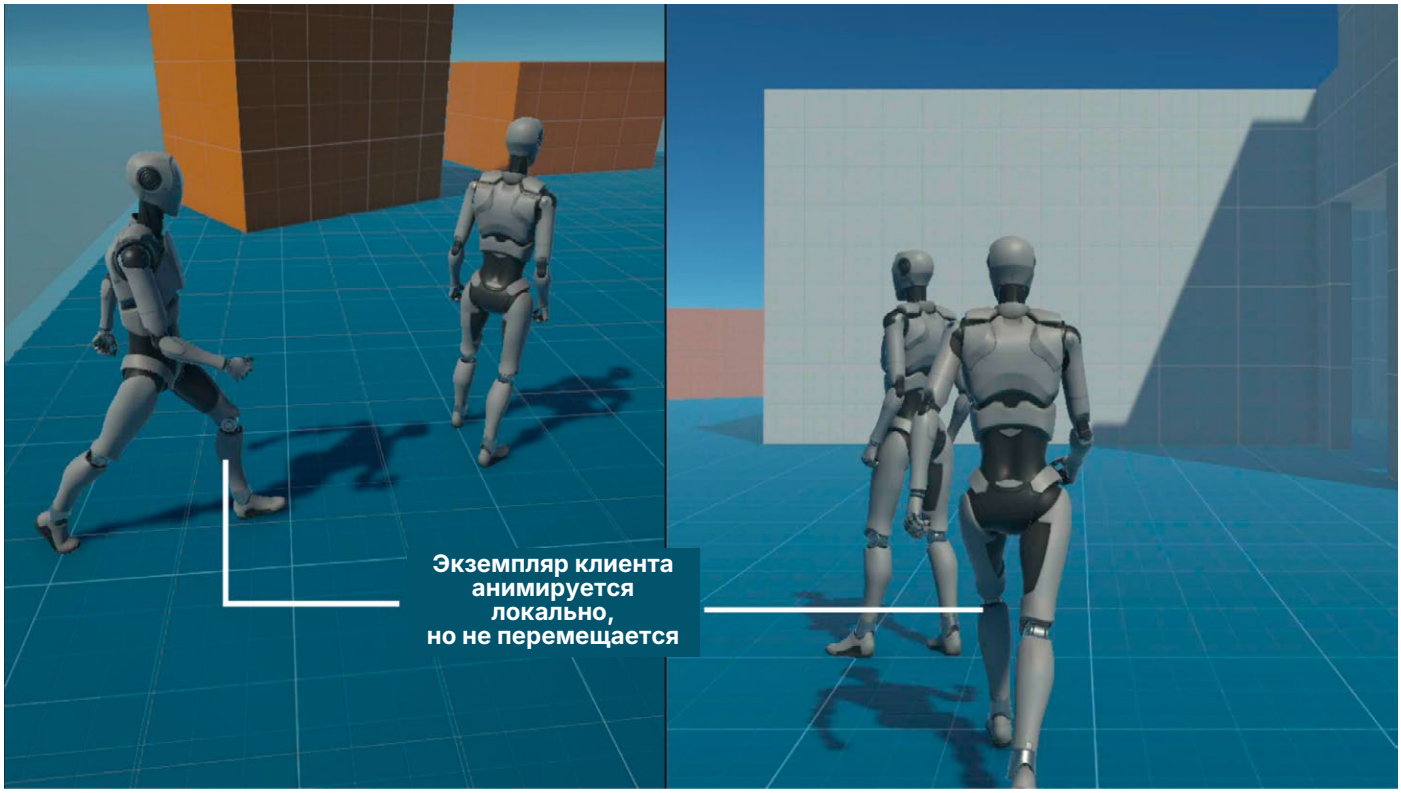


Добавьте компоненты NetworkTransform и NetworkAnimator.

Окно клиента представляет второе устройство, подключённое к хосту. В идеале любое действие на хосте должно отображаться на клиенте и наоборот.

NetworkTransform синхронизирует позицию, поворот и масштаб Transform, а NetworkAnimator - состояния анимации. Теперь перемещения игрока-хоста по сцене Playground передаются клиенту в Multiplayer Play Mode.

Однако не всё работает как ожидается. Переключитесь на окно клиента и попробуйте управлять персонажем. Хост корректно синхронизируется с клиентом, но перемещения клиента могут не отображаться на хосте.



Игрок словно бежит на месте.

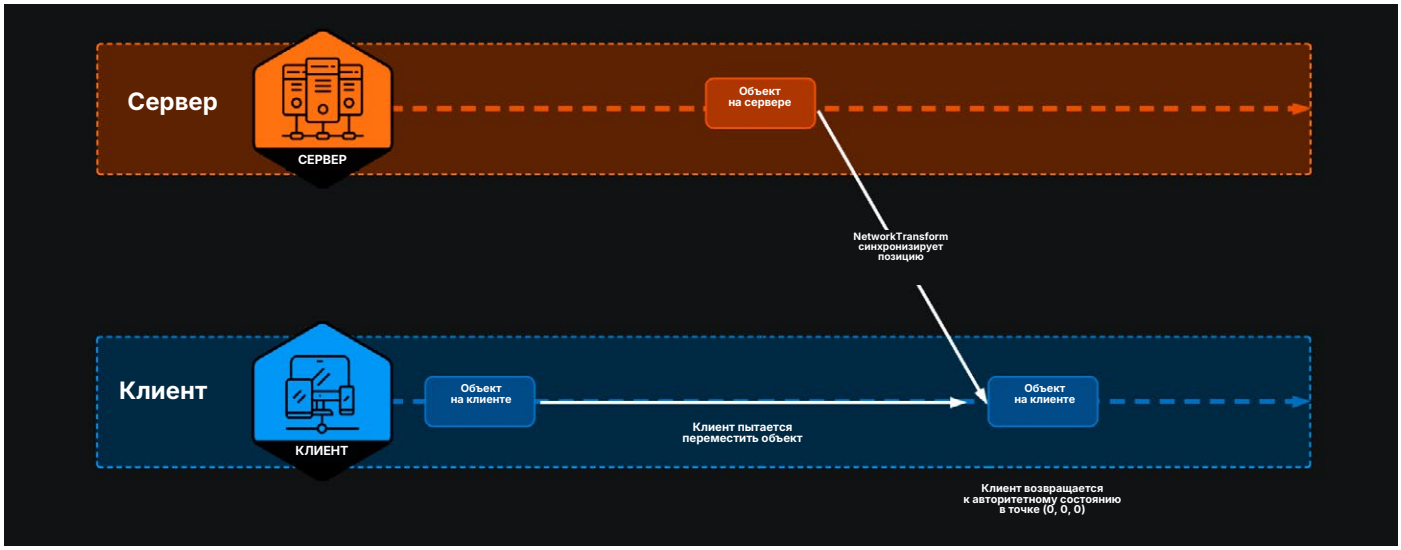
Клиент получает ввод - это видно по анимации бега на месте, - но экземпляр игрока не перемещается. Причина в том, что NetworkTransform подчиняется серверу и синхронизирует с клиентом только позицию сервера.

При попытке переместить игрока на клиенте сервер переопределяет требуемую позицию и возвращает объект в точку (0, 0, 0).

Передача полномочий клиенту

По умолчанию NetworkTransform работает в режиме серверных полномочий. Изменения осей Transform отслеживаются на сервере и передаются подключённым клиентам.

В нашем примере переместить игрока на клиенте не удаётся: сервер сохраняет авторитетное состояние с Transform в точке (0, 0, 0) и переопределяет изменения клиента.



Серверные полномочия переопределяют состояние клиента.

Один из способов решить проблему - передать полномочия от сервера клиенту. Тогда клиент сможет управлять собственным Transform, а сервер не будет переопределять изменения.

Для этого создадим компонент ClientNetworkTransform, как в следующем примере кода. Он заменяет серверные полномочия полномочиями владельца:

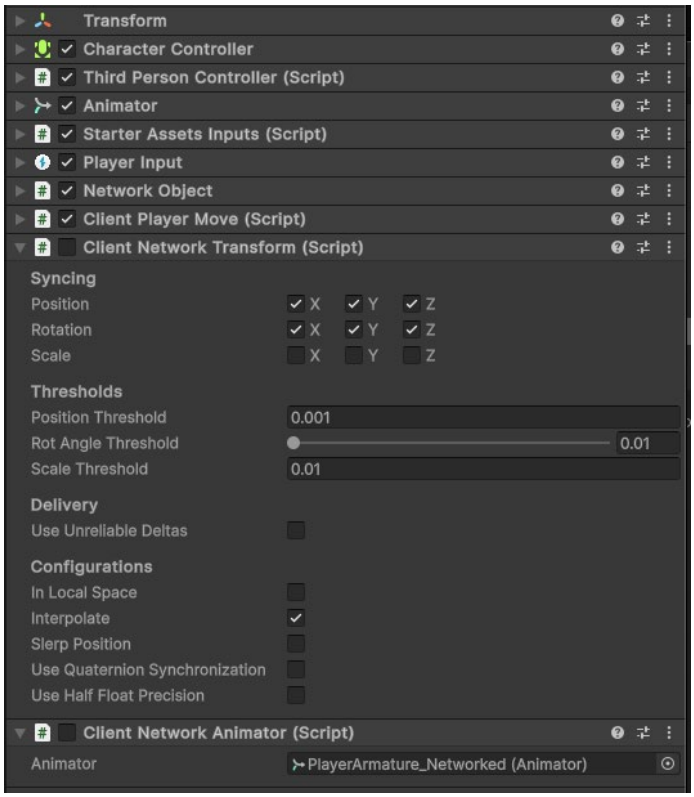
```
using Unity.Netcode.Components;
using UnityEngine;
namespace NetcodeDemo
{
[DisallowMultipleComponent]
   public class ClientNetworkTransform : NetworkTransform
   {
protected override bool OnIsServerAuthoritative()
       {
           return false;
       }
   }
}
```

Этот код переопределяет метод OnIsServerAuthoritative и возвращает false. В префабе NetworkObject игрока замените NetworkTransform пользовательским ClientNetworkTransform.

Аналогичным образом можно создать NetworkAnimator, управляемый клиентом:

```
using Unity.Netcode.Components;
using UnityEngine;
namespace NetcodeDemo
{
[DisallowMultipleComponent]
    public class ClientNetworkAnimator: NetworkAnimator
    {
protected override bool OnIsServerAuthoritative()
        {
            return false;
        }
    }
}
```

Замените NetworkAnimator компонентом ClientNetworkAnimator и не забудьте назначить поле Animator в Inspector.



Компоненты ClientNetworkTransform и ClientNetworkAnimator.

Теперь в Multiplayer Play Mode можно управлять игроком с клиента: его позиция и состояния анимации должны корректно синхронизироваться с хостом.

Поведение под управлением клиента также снижает задержку в сетевых приложениях. В режиме полномочий владельца сетевые действия выполняются сразу и ощущаются отзывчивыми: клиенту не нужно ждать, пока пакет дойдёт до сервера и вернётся обратно.

Однако применять такое поведение следует осторожно. Оно улучшает отзывчивость для игрока, но создаёт риски безопасности: режим полномочий владельца упрощает модификацию и взлом приложения, а в соревновательной онлайн-игре этим обязательно воспользуются. Для большей безопасности выбирайте серверные полномочия.

Компоненты в режиме полномочий владельца

Скрипты из примеров выше можно написать самостоятельно либо взять готовые ClientNetworkTransform и ClientNetworkAnimator из пакета Multiplayer Samples Utilities проекта Unity Boss Room (com.unity.multiplayer.samples.coop).

У этой реализации ClientNetworkTransform есть потенциальные проблемы:

- Передача владения: смена владельца не всегда проходит плавно, из-за чего объекты могут скачкообразно перемещаться или рассинхронизироваться.

- Иерархическое владение: ClientNetworkTransform нельзя использовать как дочерний объект NetworkTransform, управляемого сервером.

- Отклонение обновлений: сервер не может отклонять обновления клиента, поскольку система поддерживает только владение клиентом, но не совместное владение клиента и сервера.

- Перемещение при создании: сервер не может переместить объект сразу после его создания во владении клиента.

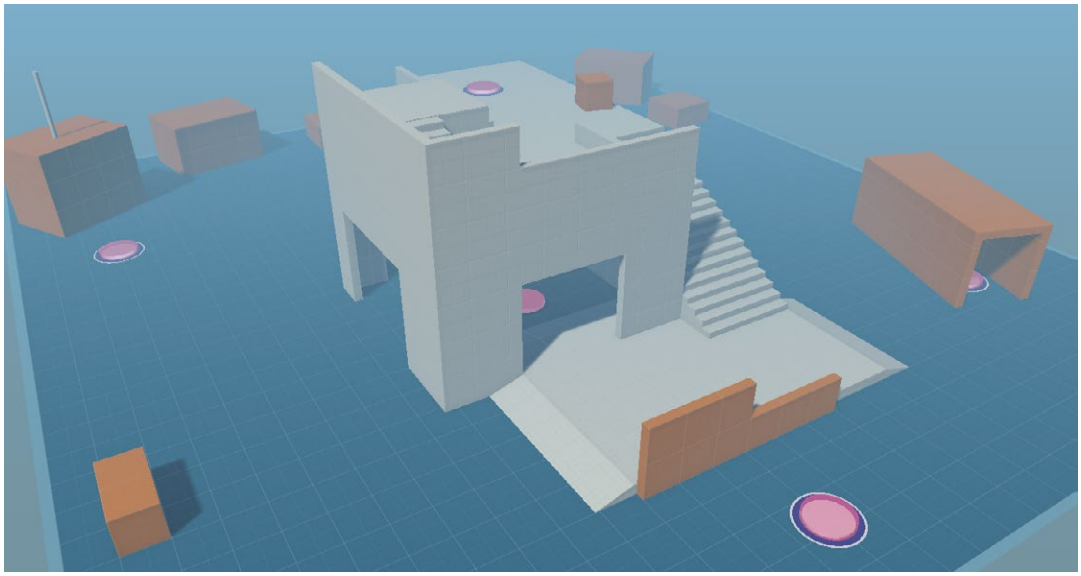
Во многих случаях ClientNetworkTransform подходит для управления Transform, принадлежащим клиенту. Однако перед внедрением учитывайте перечисленные ограничения.

Синхронизация с серверными полномочиями

Часть полномочий можно передать клиенту ради отзывчивости, однако некоторые перемещения должны выполняться только на сервере. Как правило, серверные полномочия помогают избежать дисбаланса и нечестных преимуществ, которые дают управляемые клиентом действия.

Например, если клиенты сами выбирают место появления на карте, её планировка может дать им несправедливое преимущество. Лучше, чтобы сервер случайным образом назначал одну из заранее заданных точек появления.

Для этого создайте простые визуальные объекты и расставьте их в возможных местах появления игроков.



Точки появления стратегически размещены по всему уровню.

Управлять ими может несетевой MonoBehaviour. В данном примере класс ServerPlayerSpawnPoints содержит список m_SpawnPoints со ссылками на GameObject каждой точки появления.

В примере также используется универсальный шаблон Singleton из проекта Unity для Asset Store Level up your code with design patterns and SOLID:

```
public class ServerPlayerSpawnPoints : Singleton<ServerPlayerSpawnPoints>
{
[SerializeField]
   private List<GameObject> m_SpawnPoints;
   public GameObject GetRandomSpawnPoint()
   {
       if (m_SpawnPoints.Count == 0)
           return null;
       return m_SpawnPoints[Random.Range(0, m_SpawnPoints.Count)];
   }
}
```

Затем NetworkBehaviour с именем ServerPlayerMove может использовать экземпляр ServerPlayerSpawnPoints для случайного выбора точки появления.

```
using Unity.Netcode;
using UnityEngine;
[DefaultExecutionOrder(0)] // Выполнить до ClientNetworkTransform
public class ServerPlayerMove : NetworkBehaviour
{
    public override void OnNetworkSpawn()
    {
        // Выполнять только на сервере
        if (!IsServer)
        {
            enabled = false;
            return;
        }
        SpawnPlayer();
        base.OnNetworkSpawn();
    }

    // Перейти к следующей свободной точке при появлении
    void SpawnPlayer()
    {
        var spawnPoint = ServerPlayerSpawnPoints.Instance.GetRandomSpawnPoint();
        var spawnPosition = spawnPoint ? spawnPoint.transform.position : Vector3.zero;
        transform.position = spawnPosition;
    }
}
```

Вся логика выполняется в OnNetworkSpawn. При каждом подключении клиента вызов SpawnPlayer помещает игрока в случайно выбранную точку появления. Проверка IsServer гарантирует, что это происходит только на сервере, который поддерживает авторитетное состояние игры.

Добавьте скрипт ServerPlayerMove в префаб PlayerArmature_Networked. В Multiplayer Play Mode каждый клиент подключится и появится в случайной точке сцены Playground.

Этот пример показывает, как NetworkBehaviour взаимодействует с элементами сцены, не управляемыми по сети. Он использует статические данные и объекты, заранее настроенные в Hierarchy.

При подключении клиента ServerPlayerMove достаточно получить одну случайную точку появления из существующей игровой сцены. Это сокращает объём передаваемых по сети данных.

Игрок появляется в случайной точке.

Несколько важных моментов:

- Поскольку ClientNetworkTransform использует полномочия владельца, компонент CharacterController необходимо отключить в Awake. Снова включите CharacterController после того, как ServerPlayerMove задаст позицию игрока, иначе компонент переопределит рассчитанные координаты и вернёт объект в центр мира.

- Также заполните список m_SpawnPoints в Inspector, чтобы игроки не появлялись в точке (0, 0, 0).

- Задайте атрибуту DefaultExecutionOrder меньшее значение, чтобы ServerPlayerMove выполнялся раньше ClientPlayerMove. Например, [DefaultExecutionOrder(0)] повышает приоритет ServerPlayerMove и позволяет ему запуститься первым.

Теперь к хосту нашего многопользовательского проекта могут подключаться несколько клиентов. Персонажи от третьего лица появляются в заданных местах уровня и синхронизируют перемещение и анимацию в реальном времени.

Такая синхронизация необходима для многопользовательской игры. Компоненты NetworkTransform и NetworkAnimator обеспечивают её из коробки, но игровая логика также потребует собственных NetworkBehaviour.

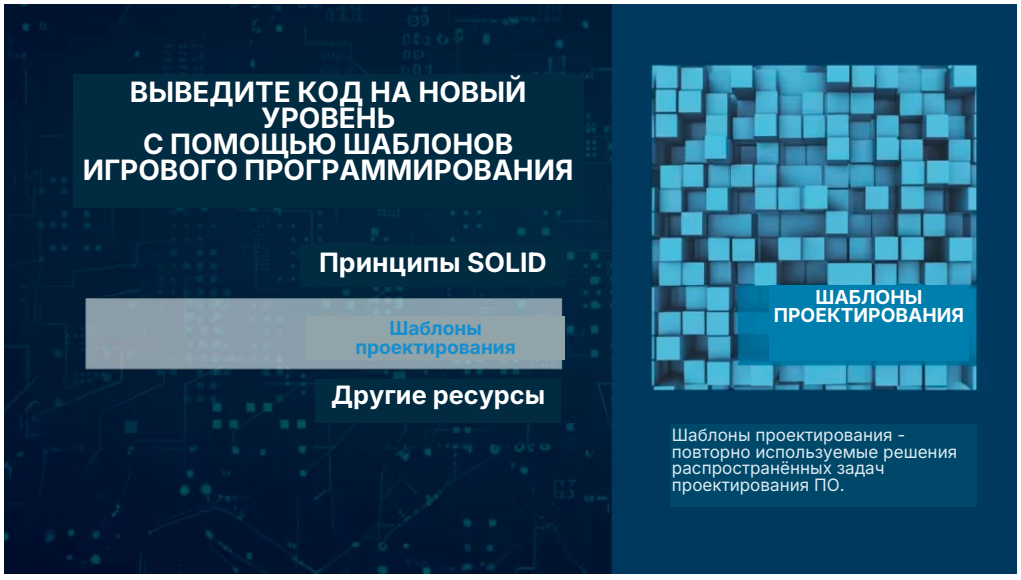
Теперь рассмотрим другие способы синхронизации данных и игровых состояний по сети.

Шаблон проектирования Singleton

Singleton предоставляет удобный доступ к единственному экземпляру определённого типа во время выполнения. Однако он создаёт дополнительные зависимости, поэтому учитывайте недостатки этого шаблона.

В Netcode for GameObjects Singleton используется при каждом обращении к NetworkManager.Singleton. В примере проекта также есть универсальная реализация Singleton для любого типа MonoBehaviour.

Подробнее о Singleton рассказывает электронная книга Level up your code with design patterns and SOLID. В ней также показаны альтернативные способы взаимодействия объектов сцены, например обычные события и каналы событий.



Скачайте бесплатную электронную книгу Unity о шаблонах проектирования. Другие углублённые руководства для программистов, технических художников, художников и дизайнеров собраны в центре лучших практик Unity.

Сетевая синхронизация

Сетевая синхронизация необходима для согласованного и честного игрового процесса для всех игроков.

Вы уже синхронизировали перемещение и анимацию NetworkObject игрока с помощью модели под управлением клиента. Но игровой процесс редко ограничивается персонажем: игрок может стрелять, открывать двери и взаимодействовать с объектами сцены. Такие взаимодействия тоже должны работать по сети, а их состояния - синхронизироваться между клиентами и сервером.

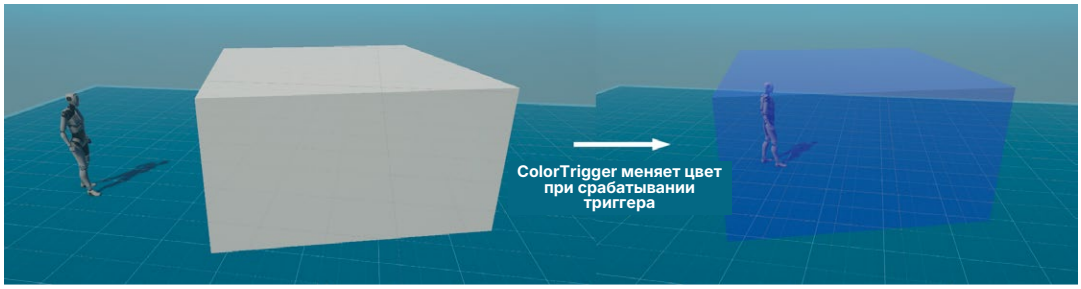
В этом разделе мы настроим клиент-серверный обмен для взаимодействия игрока с окружением. Для этого реализуем сетевые состояния игры и отправку удалённых вызовов процедур (RPC) на сервер и обратно.

Игровая механика

Рассмотрим клиент-серверный обмен на примере простой игровой механики.

Добавим на уровень триггерные коллайдеры, которые меняют цвет при контакте с игроком: при касании одного игрока объект принимает один цвет, а при касании другого - другой.

Создайте в сцене новый GameObject, например куб. Добавьте BoxCollider и включите параметр IsTrigger. Создайте прозрачный материал, назначьте его компоненту MeshRenderer - в примере используется шейдер URP/Lit - и задайте нейтральный исходный цвет, например белый.



Триггер будет получать сетевое значение цвета.

Теперь добавим сетевую синхронизацию. Для передачи изменения цвета по сети используем NetworkVariable и RPC.

Объявление NetworkVariable

NetworkVariable - специализированная переменная для синхронного управления состоянием по сети. Изменения на сервере передаются всем клиентам. Такой механизм подходит для непрерывно синхронизируемых данных: позиции, здоровья или, как в этом примере, цвета триггера.

Создадим NetworkBehaviour с именем ColorTrigger и добавим его к объекту триггера. Скрипт будет содержать NetworkVariable m_NetworkColor со значением типа Color.

```
using UnityEngine;
using Unity.Netcode;
public class ColorTrigger : NetworkBehaviour
{
   public NetworkVariable<Color> m_NetworkColor = new NetworkVariable<Color>(Color.
white);
   private Material m_InstanceMaterial;

   public override void OnNetworkSpawn()
   {
       m_NetworkColor.OnValueChanged += OnColorChanged;
       MeshRenderer meshRenderer = GetComponent<MeshRenderer>();
       if (meshRenderer != null)
       {
           m_InstanceMaterial = new Material(meshRenderer.material);
           meshRenderer.material = m_InstanceMaterial;
           UpdateMaterialColor(m_NetworkColor.Value);
       }
   }
```

```
    public override void OnNetworkDespawn()
    {
        m_NetworkColor.OnValueChanged -= OnColorChanged;
    }
    private void OnColorChanged(Color oldColor, Color newColor)
    {
        UpdateMaterialColor(newColor);
    }

    private void UpdateMaterialColor(Color newColor)
    {
        if (m_InstanceMaterial != null)
        {
            m_InstanceMaterial.SetColor("_BaseColor", newColor);
        }
    }
}
```

Когда игрок входит в триггер, скрипт переключает свойство базового цвета экземпляра материала. Для этого используется NetworkVariable m_NetworkColor.

Разберём принцип работы:

- NetworkVariable хранит фактическое значение цвета и синхронизирует его между всеми клиентами. Скрипт выполняется и на сервере, и на клиентах, но по умолчанию записывать NetworkVariable может только сервер; клиенты могут лишь читать значение Color.

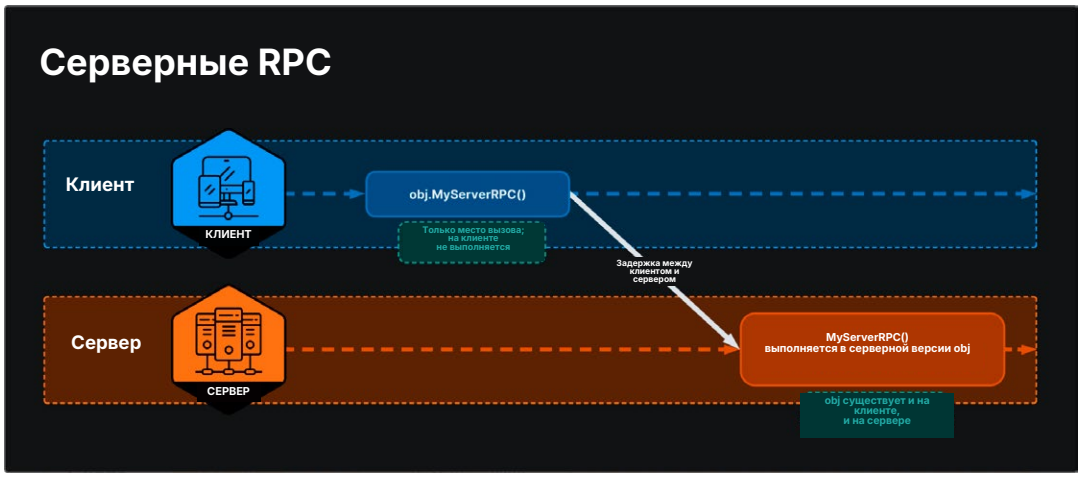
- Событие OnValueChanged обновляет базовый цвет материала триггера при каждом изменении NetworkVariable. Скрипт подписывается на событие в OnNetworkSpawn и отписывается в OnNetworkDespawn.

NetworkTransform предназначен именно для данных Transform - позиции, поворота и масштаба, - тогда как NetworkVariable может синхронизировать более общие данные: примитивные типы, пользовательские структуры и другие значения игрового состояния.

На клиенте нельзя напрямую изменить NetworkVariable, поскольку право записи принадлежит серверу. Клиент должен запросить изменение у сервера; сервер обновляет состояние и передаёт его обратно, после чего клиент видит результат.

Для такого обмена используется RPC. Удалённый вызов процедур позволяет одному устройству потребовать от другого выполнить действие или обновление. RPC можно отправлять с клиента на сервер и в обратном направлении.

Рассмотрим, как реализовать RPC для этой задачи.



Серверный RPC удалённо вызывается клиентом и выполняется на сервере.

Добавление RPC

RPC позволяют удалённо вызывать функции на сервере или других клиентах. Метод с атрибутом [Rpc(SendTo.Server)] клиент вызывает на сервере, а метод с [Rpc(SendTo.Client)] сервер вызывает на клиентах. Также поддерживается устаревший синтаксис [ServerRpc] и [ClientRpc] для серверных и клиентских RPC.

RPC похожи на обычные методы, но должны соответствовать нескольким соглашениям:

- Атрибут Rpc: пометьте метод атрибутом [Rpc] и укажите получателя. Например, [Rpc(SendTo.Server)] вызывает метод только на сервере.

- Именование: имя метода должно оканчиваться суффиксом Rpc, например DoSomethingRpc.

RPC лучше подходят для отдельных событий - действий игрока или изменений игрового состояния, которые не требуют непрерывной синхронизации.

Добавьте следующие методы в скрипт TriggerColor:

```
private void OnTriggerEnter(Collider other)
 {
     NetworkObject networkObject = other.GetComponent<NetworkObject>();
     if (IsClient && networkObject != null && networkObject.IsOwner)
     {
         ChangeColorServerRpc(networkObject.OwnerClientId);
     }
 }
```

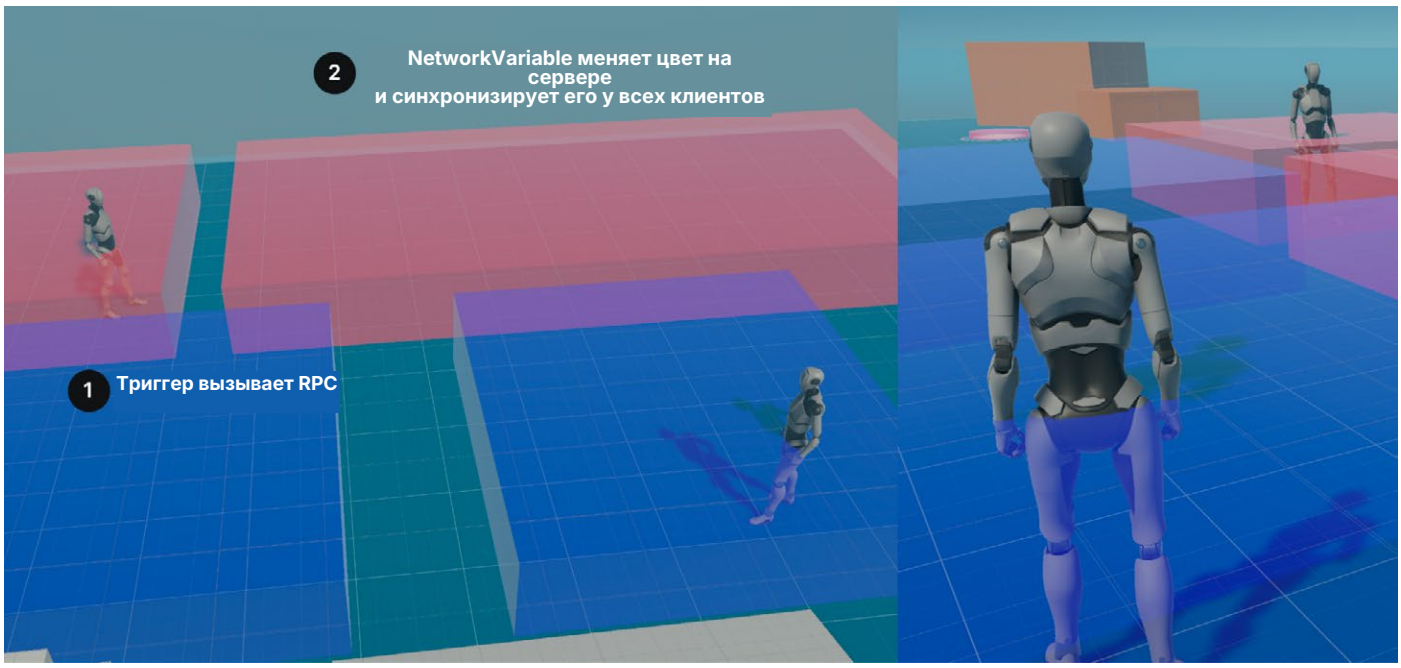
```
  [Rpc(SendTo.Server)]
private void ChangeColorServerRpc(ulong playerId)
{
    // Простая система команд: синий для чётных, красный для нечётных
    Color newColor =
        (playerId % 2 == 0) ? new Color(0, 0, 1, 0.5f) : new Color(1, 0, 0, 0.5f);

    m_NetworkColor.Value = newColor;
}
```

В одиночной игре нужную логику можно добавить в OnTriggerEnter. В многопользовательской игре взаимодействия обычно обрабатываются через клиент-серверный обмен, чтобы состояния всех клиентов оставались согласованными.

Вместо прямого изменения m_NetworkColor метод OnTriggerEnter проверяет, является ли текущий экземпляр игры клиентом. Если да, он вызывает ChangeColorServerRpc и передаёт OwnerClientId.

Метод выбирает новый цвет по идентификатору игрока - в этом примере чётным номерам соответствует синий, нечётным красный - и обновляет m_NetworkColor.

Изменение передаётся всем подключённым клиентам, поэтому каждый экземпляр игры видит одинаковое состояние. Когда m_NetworkColor меняется, на всех клиентах вызывается OnColorChanged и обновляет цвет материала триггера.



Эта простая механика демонстрирует клиент-серверное взаимодействие.

Механика триггера

Пример прост, но похожая механика встречается во многих многопользовательских играх, где действия игрока вызывают визуальные изменения окружения. Например:

- В кооперативной головоломке игроки могут воздействовать на окружение кнопками и триггерами. Визуальная реакция игрового мира служит общим сигналом и помогает согласовать действия.

- В соревновательном шутере игроки захватывают контрольные точки, которые часто окрашиваются в цвет владеющей ими команды. Этот визуальный сигнал помогает игрокам корректировать стратегию.

- В сетевых RPG отдельные области уровня могут накладывать усиления, ослабления и другие эффекты.

RPC и NetworkVariable

Выбор между NetworkVariable и RPC при синхронизации данных зависит от задачи:

- NetworkVariable и NetworkTransform подходят для непрерывно синхронизируемых данных: позиции, здоровья или, как здесь, цвета триггера. В этом примере NetworkVariable хранит активный цвет триггера.

    - Примеры: здоровье, позиционные данные, игровой счёт.

- Удалённые вызовы процедур (RPC) лучше подходят для отдельных событий, например действий игрока или изменений состояния, не требующих непрерывной синхронизации. В этом примере RPC сообщает серверу, что игрок вошёл в зону триггера, после чего сервер меняет цвет с учётом идентификатора игрока.

    - Примеры: действия игрока - выстрел или применение способности; игровые события - создание объекта, начало игры или победа.

Оба механизма необходимы для синхронизации сетевой игры: NetworkVariable управляет длительными состояниями, а RPC - отдельными событиями и действиями.

Следующие примеры помогут определить, когда применять NetworkVariable, а когда RPC.

Задача / система	RPC	NetworkVariables
Управление инвентарём	Уведомляют клиентов о подборе или использовании предмета, его добавлении и удалении.	Хранят инвентарь каждого игрока; NetworkList синхронизирует список предметов.
Боевая система	Выполняют атаки и особые приёмы; RPC может наносить урон и эффекты.	Синхронизируют здоровье, состояние игрока и активные эффекты.
Взаимодействие с окружением	Запускают действия: открытие двери или включение механизма.	Хранят состояние интерактивных объектов, например открыта ли дверь.
Цели	Сообщают о выполнении цели.	Отслеживают прогресс, число собранных предметов и выполненных задач.

Подробнее см. на странице документации RPC vs NetworkVariable.

Проектирование многопользовательской игры

Освоив основы Netcode, при разработке важно мыслить категориями сетевой многопользовательской игры. Поведение, простое в одиночной игре, часто усложняется, когда его необходимо воспроизвести на нескольких устройствах с помощью NetworkVariable или RPC.

Заранее спланируйте синхронизацию состояний между клиентами и сервером. Для постоянно обновляемых данных используйте NetworkVariable, для отдельных событий - RPC. Передавайте только необходимые данные, чтобы сократить сетевой трафик.

Одиночную игру можно преобразовать в многопользовательскую, но обычно эффективнее учитывать сетевой режим с самого начала. Заранее решите, какие объекты и действия контролирует сервер, а какие - клиенты. Серверные полномочия дают максимальную безопасность и согласованность, но для снижения задержки некоторых действий их следует разумно сочетать с полномочиями клиента.

Задержка часто становится серьёзной проблемой при создании сетевой игры. Далее рассмотрим её влияние на многопользовательский процесс.

Для вдохновения посмотрите доклад Unite 2024 о наиболее распространённых и серьёзных ошибках при создании многопользовательских игр и способах их избежать.

Задержка и производительность сети

Если вы играли онлайн, то наверняка замечали, как задержка ухудшает впечатление от игры. Недостаточная пропускная способность и нестабильное соединение приводят к рывкам персонажей, неравномерной частоте кадров и заметной задержке ввода.

Теперь, понимая обмен между клиентами и сервером, легко увидеть причину. При передаче через интернет многое может пойти не так. Даже с механизмами надёжности Unity Transport UDP-пакеты способны потеряться, прийти в неверном порядке или повредиться по пути. Всё это вызывает задержку, которую игрок воспринимает как лаг.

Имитация задержки

Чтобы изучить и смягчить влияние задержки во время разработки, в Unity Editor можно имитировать разные сетевые условия с помощью Debug Simulator компонента UnityTransport или Network Simulator.

Debug Simulator в Unity Transport

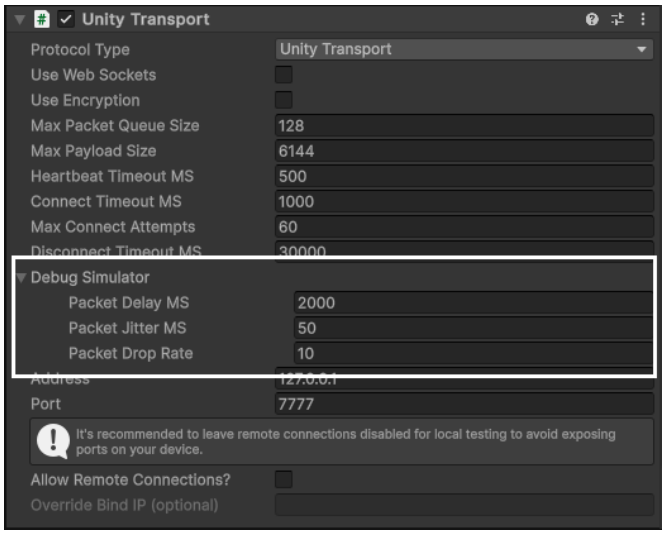
Настройте Debug Simulator в компоненте UnityTransport, чтобы добавить искусственную задержку, джиттер и потерю пакетов. Эти параметры находятся в Inspector объекта NetworkTransport сцены. Задайте несколько значений, имитирующих перегрузку сети:

Latency: добавляет задержку в миллисекундах для имитации медленного соединения. Например, значение 100 мс воспроизводит умеренную сетевую задержку.

Jitter: добавляет колебания задержки, имитируя нестабильные сетевые условия. Например, задайте 50 мс и посмотрите, как нестабильное соединение влияет на игру.

Packet Loss: задаёт процент отбрасываемых пакетов для имитации плохого качества связи. Попробуйте установить потерю пакетов 5 %, чтобы оценить её влияние на игровой процесс.

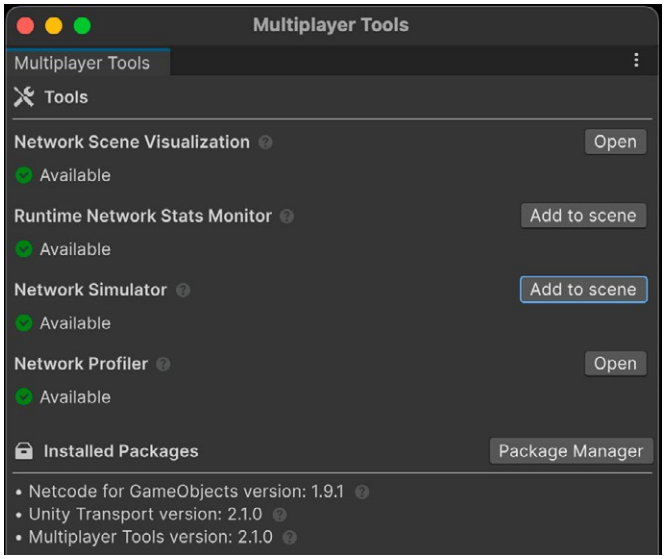
Снова запустите игру и посмотрите, как смоделированные условия влияют на игровой процесс.



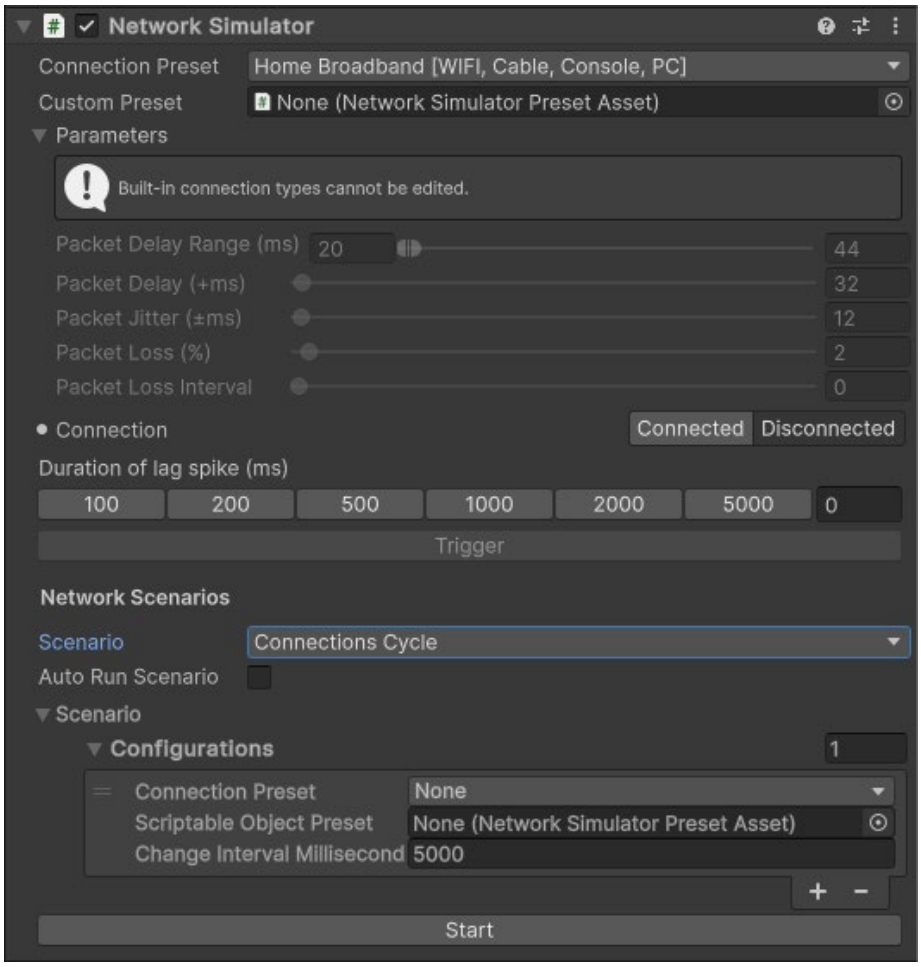
Настройте Debug Simulator в Unity Transport.

Network Simulator

Network Simulator из пакета Multiplayer Tools позволяет тестировать неблагоприятные сетевые условия и находить проблемы до выпуска игры. Он имитирует сетевые события: разрывы соединения, всплески задержки и потерю пакетов.



Установите Network Simulator из пакета Multiplayer Tools.



Проверьте влияние задержки с помощью Network Simulator.

Другие инструменты эмуляции сетевых условий

Debug Simulator и Network Simulator работают только в редакторе. Для сборок приложения задержку можно имитировать с помощью других инструментов эмуляции сетевых условий. Например, clumsy для Windows и Network Link Conditioner для macOS/iOS позволяют воспроизводить различные сетевые условия и тщательно тестировать игру.

Подробнее см. в разделе «Тестирование и отладка» далее в этом руководстве.

Учет влияния задержки на производительность приложения - одна из главных задач при разработке многопользовательских игр. К счастью, существует ряд подходов, позволяющих ослабить ее последствия. Рассмотрим некоторые из них.

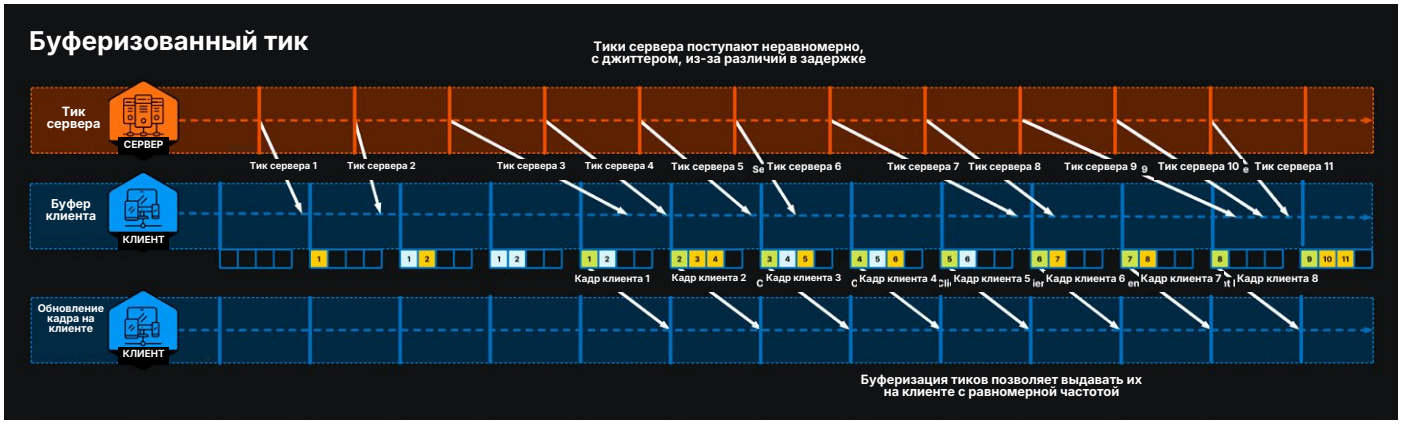
Интерполяция на стороне клиента

Один из способов уменьшить влияние задержки - интерполяция на стороне клиента. При таком подходе клиент не отображает состояние сразу, а намеренно откладывает отрисовку на небольшой интервал интерполяции.

В клиент-серверной топологии клиент обычно отображает состояние, отстающее от серверного примерно на половину времени кругового прохождения сигнала (RTT). Интерполяция на стороне клиента добавляет к этому еще одну намеренную задержку.

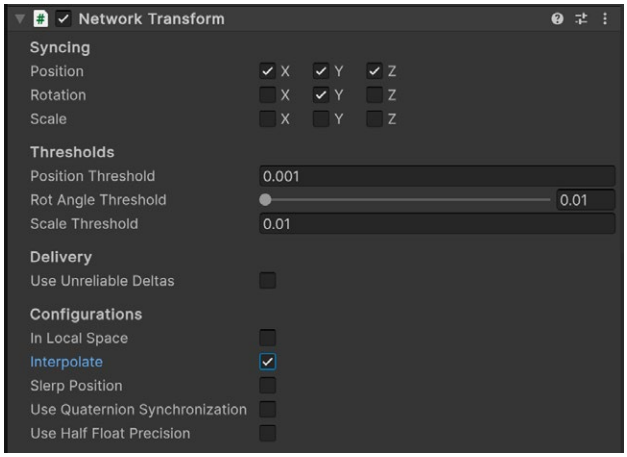
Работая с небольшим отставанием, клиент может буферизовать поступающие с сервера обновления состояния. Когда приходит время отобразить следующее обновление, клиент вычисляет интерполированное состояние по двум последним серверным тикам.

Буфер позволяет клиенту выводить обновления равномерно, даже если серверные тики поступают нерегулярно из-за джиттера. Интерполированные состояния скрывают небольшую задержку и дрожание времени доставки.



Клиент отображает интерполированные состояния из буфера.

В Netcode for GameObjects интерполяция на стороне клиента включается параметром компонента NetworkTransform. Параметр Interpolation интерполирует положение, поворот и масштаб соответствующего GameObject.



Интерполяция на стороне клиента включена в NetworkTransform.

Учтите, что интерполяция на стороне клиента неизбежно добавляет небольшую задержку отрисовки.

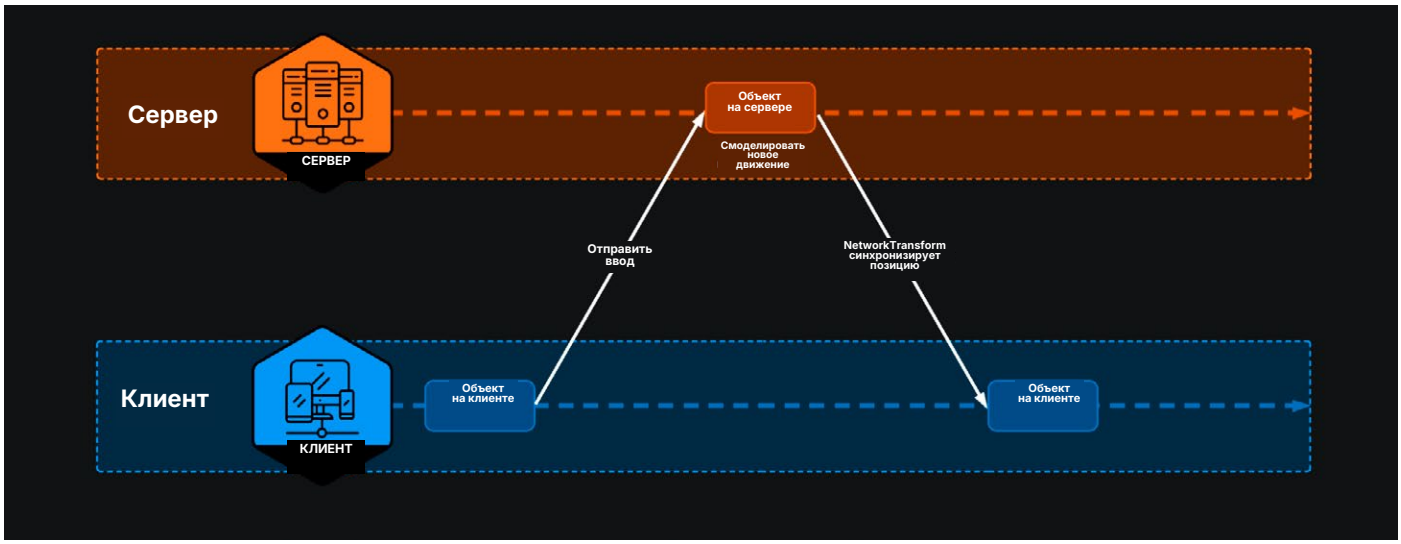
Прогнозирование и упреждение на стороне клиента

В предыдущем примере ClientNetworkTransform передает клиенту непосредственное управление перемещением игрока, поэтому игра ощущается более отзывчивой. Однако во многих играх клиентская авторитетность недопустима из-за требований игрового дизайна, честности и безопасности.

Зачем нужен авторитетный сервер

В соревновательных играх, где особенно важна честность, клиентская авторитетность позволяет игрокам жульничать и использовать уязвимости, изменяя клиентский код или данные. Играм со сложным взаимодействием игроков и общим игровым миром часто нужен авторитетный сервер: он обеспечивает целостность данных, препятствует вмешательству и поддерживает единообразный игровой процесс для всех участников. Такой подход сохраняет равные условия и целостность игры.

Поэтому вместо ClientNetworkTransform, где объектом управляет владелец, используется стандартный компонент NetworkTransform. В этом случае ввод клиента не управляет игроком напрямую - управление остается только у сервера.



Перемещение NetworkTransform под управлением авторитетного сервера.

Однако авторитетность сервера может усиливать влияние задержки. Клиент не управляет персонажем на экране напрямую, а отправляет ввод на сервер. Сервер обрабатывает его, выполняет симуляцию и вычисляет состояние игры. Клиент может показать следующий кадр только после получения результатов этой симуляции.

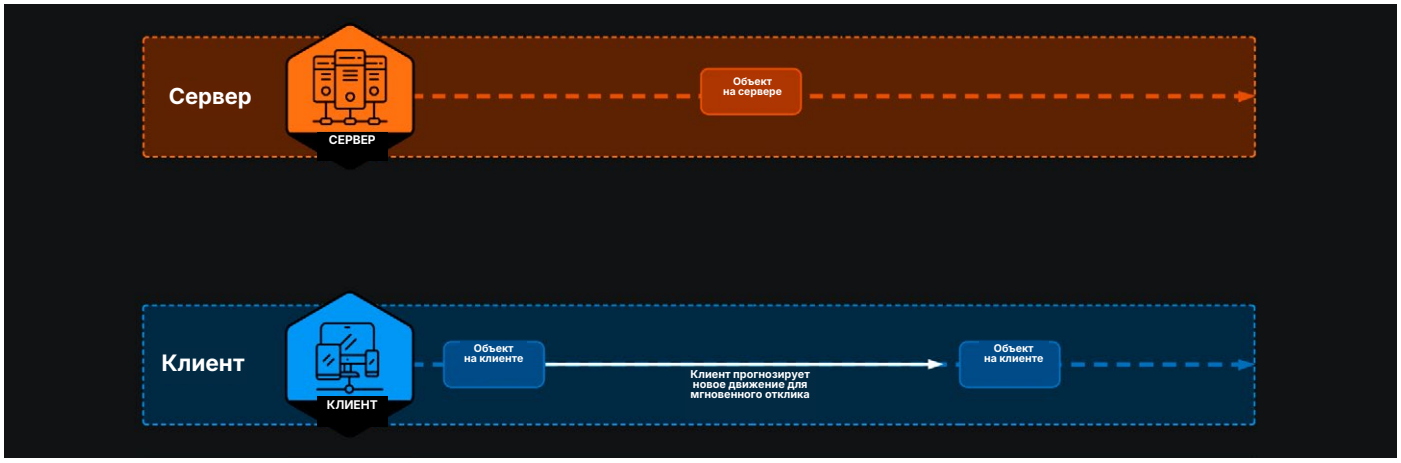
При использовании авторитетного сервера игрок замечает задержку, пока данные идут от клиента к серверу и обратно. Поэтому клиент обычно отстает от сервера, особенно когда UDP-пакеты передаются через интернет.

Для многих элементов игры такое отставание приемлемо, но в случае персонажа игрока оно может испортить ощущение от управления и затруднить игру.

Как работает прогнозирование на стороне клиента

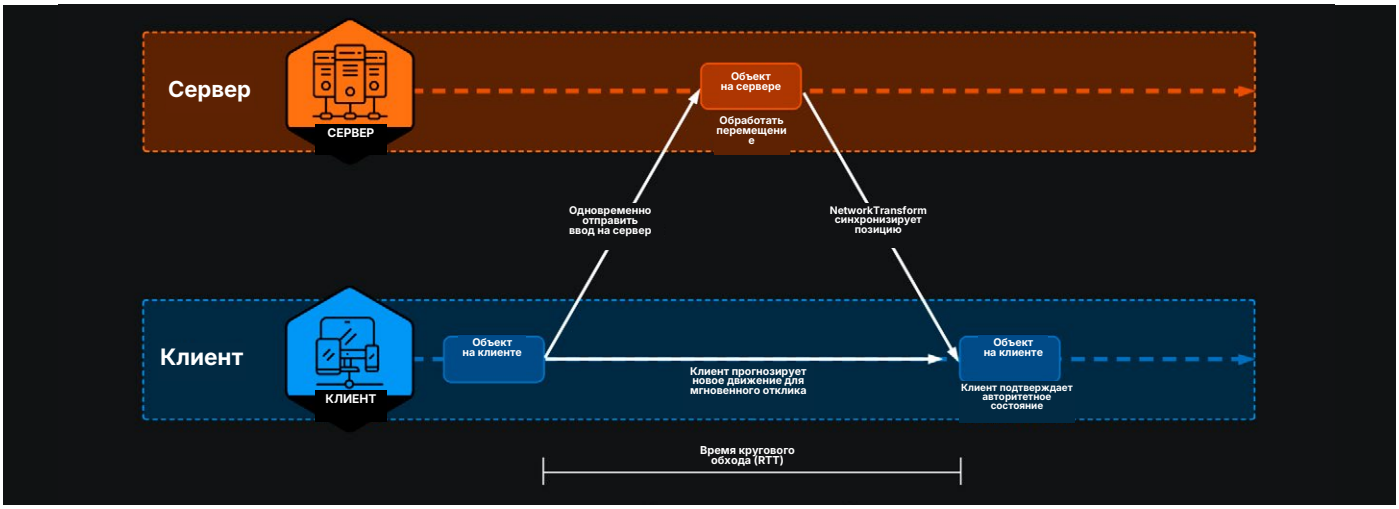
Прогнозирование на стороне клиента помогает компенсировать задержку, возникающую при серверной авторитетности. Не дожидаясь ответа сервера, клиент прогнозирует состояние игры на долю секунды вперед и сразу обновляет изображение. Подход называется прогнозированием, потому что клиент еще не знает фактическое состояние, рассчитанное сервером.

Благодаря этому действия игрока немедленно получают визуальный отклик, и управление ощущается отзывчивым.



Клиент прогнозирует состояние игры.

Одновременно клиент отправляет действия игрока на сервер в пакете. Сервер получает пакет с вводом, обрабатывает данные и выполняет симуляцию, формируя эталонное авторитетное состояние игры. Затем сервер отправляет это состояние клиенту.

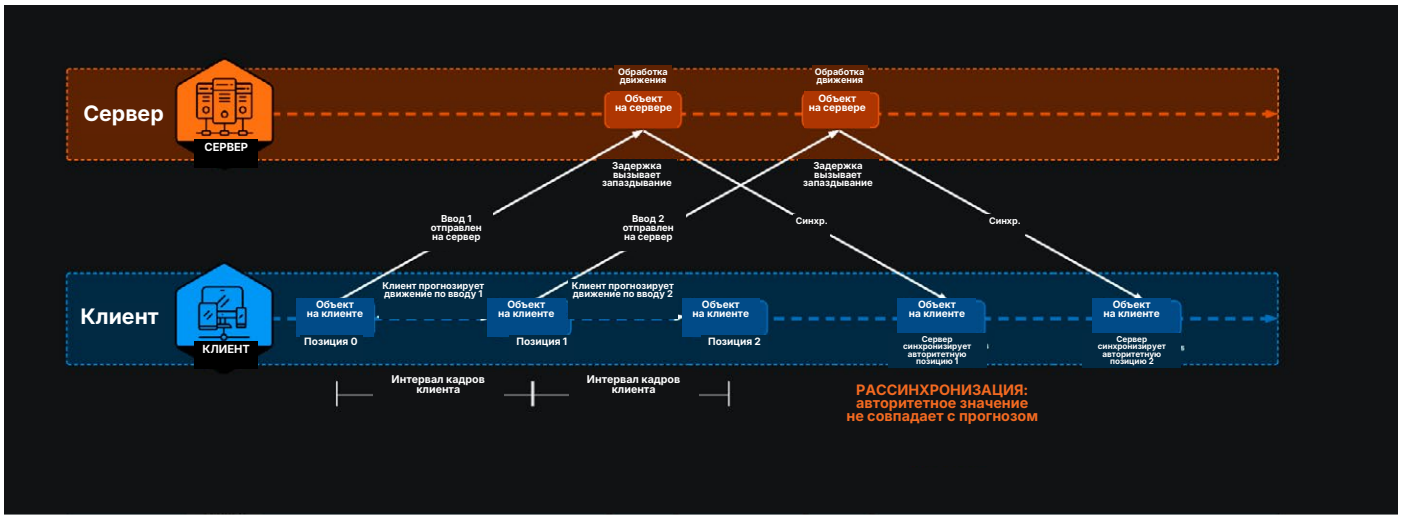


Сервер возвращает авторитетное состояние.

Согласование и откат

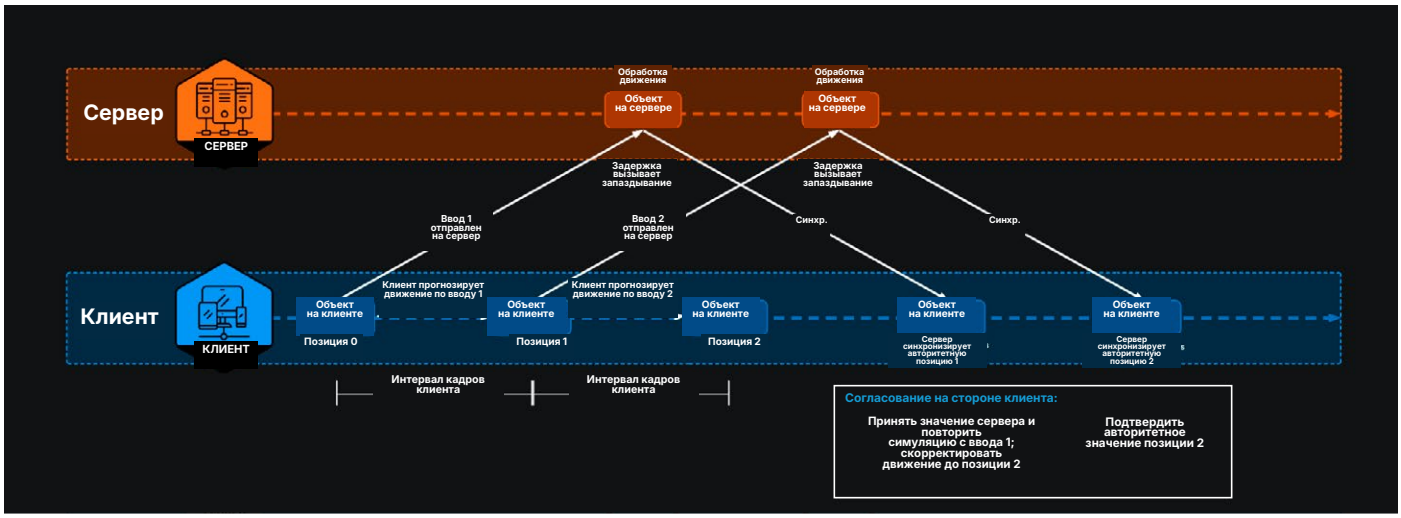
Затем клиент сравнивает авторитетное состояние с упрежденным и ищет различия. Если состояния достаточно близки, ничего исправлять не нужно и игра продолжается.

При существенном расхождении, которое также называют рассинхронизацией, клиент должен скорректировать свое состояние в соответствии с авторитетным состоянием сервера. Этот процесс называется согласованием.



Клиент получает несовпадающее состояние - происходит рассинхронизация.

Чтобы компенсировать задержку, клиент хранит историю ввода и прогнозируемых состояний за определенное число кадров. При рассинхронизации он откатывается к последнему корректному состоянию, полученному от сервера, а затем повторно выполняет симуляцию с правильными входными данными. Так состояние клиента снова синхронизируется с сервером.



Клиент устраняет рассинхронизацию путем согласования.

В одном кадре клиенту иногда приходится повторно просчитывать сразу несколько кадров, поэтому сетевое приложение требует больше вычислительных ресурсов. Однако согласование и откат позволяют сохранить плавность и целостность игрового процесса даже при сетевой задержке.

Упреждение на стороне клиента в Netcode for GameObjects

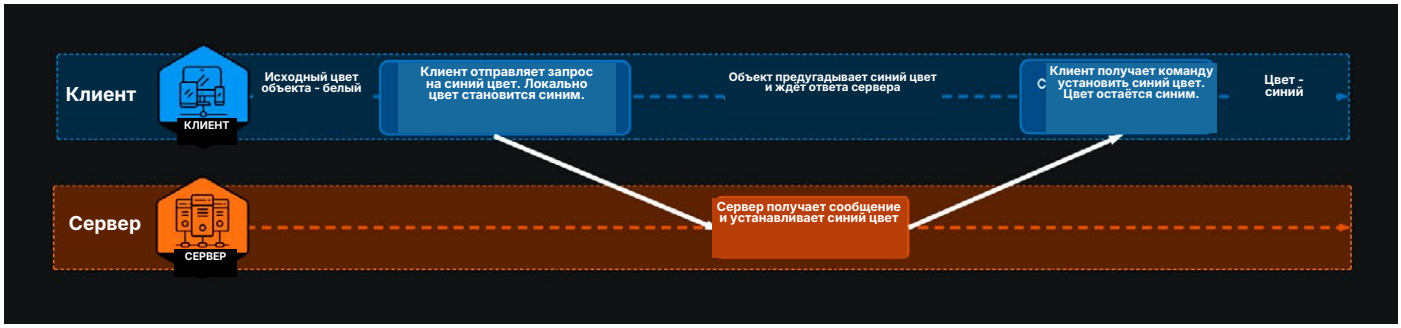
Netcode for GameObjects поддерживает клиентское упреждение - упрощенный способ работы с задержкой без полноценного прогнозирования и согласования на стороне клиента. Для этого используются следующие компоненты:

- AnticipatedNetworkVariable<T> - обобщенный компонент для скалярных и простых типов данных: целых чисел, чисел с плавающей запятой, цветов и т. п. Он подходит для свойств, не относящихся к Transform, например здоровья, счета или состояния предмета.

- AnticipatedNetworkTransform работает аналогично AnticipatedNetworkVariable, но предназначен для данных Transform: положения, поворота и масштаба.

Клиентское упреждение дает игроку немедленный визуальный отклик, пока клиент ожидает авторитетное обновление от сервера. Благодаря этому игра ощущается более отзывчивой. Например, в простом примере ColorTrigger клиент может сразу показать изменение цвета объекта с белого на синий, не дожидаясь подтверждения сервера.

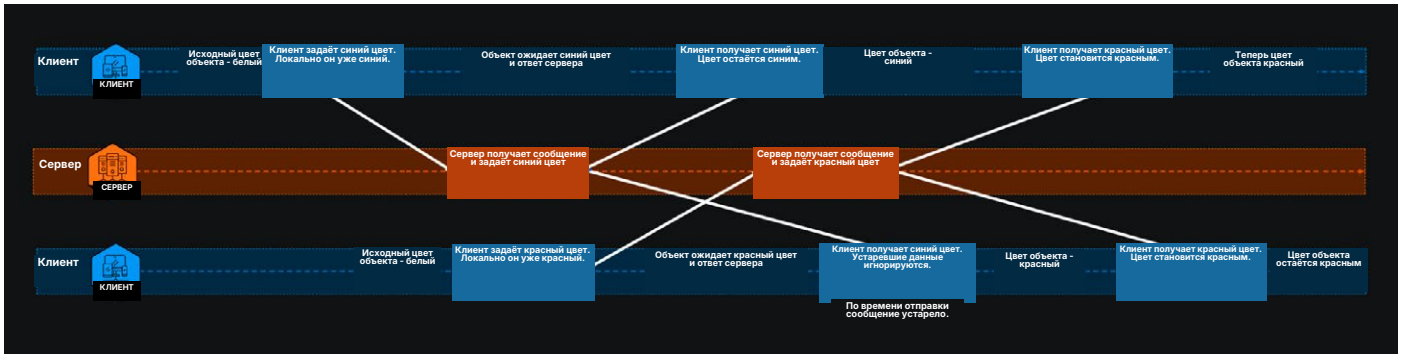
Взаимодействие клиента и сервера при упреждении выглядит примерно так:



Клиент упреждает состояние игры.

AnticipatedNetworkVariable<T> и AnticipatedNetworkTransform разделяют значения на упрежденное (визуальное) и авторитетное состояния.

Упреждение - это прогнозируемое клиентом состояние, которое обеспечивает игроку немедленный локальный отклик. Авторитетное состояние определяет сервер. Получив обновление сервера, клиент сравнивает оба состояния. Если они заметно расходятся, клиент корректирует свое состояние по серверному, сохраняя согласованность между всеми клиентами.



Клиентское упреждение может игнорировать устаревшие данные.

При клиентском упреждении необходимо учитывать «устаревшие данные» - обновления сервера, отражающие действия, которые произошли до последнего запроса клиента. В Netcode for GameObjects свойство StaleDataHandling предлагает два варианта обработки:

- StaleDataHandling.Ignore игнорирует устаревшие данные и сохраняет упрежденное значение. Это полезно, если состояние меняется быстро и изображение начинает мерцать.

- StaleDataHandling.Reanticipate обрабатывает устаревшие данные как обычное обновление сервера: выполняет откат и повторное упреждение, воспроизводя ввод игрока для сохранения согласованности.

Чтобы избежать резких визуальных скачков при расхождении серверных и упрежденных значений, используйте доступный обоим компонентам метод Smooth.

Метод Smooth принимает начальное и конечное значения, а также длительность сглаживания. Это помогает сохранить плавный и отзывчивый визуальный отклик при сетевой задержке.

Клиентское упреждение повышает отзывчивость игры, однако этот упрощенный подход охватывает не все случаи задержки и сетевых неполадок.

Обратите внимание: для полноценного отката и согласования нужна детерминированная физическая система, в которой одинаковые входные данные всегда дают одинаковый результат (см. ниже). Только так можно точно откатывать и повторно просчитывать состояния игры. Без детерминизма симуляции клиента и сервера могут расходиться.

Netcode for GameObjects не поддерживает детерминированную физику, необходимую для полноценного клиентского прогнозирования и компенсации задержки. Вместо этого пакет предлагает более простое решение на основе клиентского упреждения и сглаживания, которое повышает отзывчивость без полной системы отката.

Детерминированная физика

Детерминированная физическая система гарантирует, что при одинаковых начальных условиях и входных данных симуляция всегда даст одинаковый результат. Netcode for GameObjects использует встроенный физический движок Unity, который не является детерминированным: повторный запуск симуляции с теми же входными данными не гарантирует идентичного результата.

Netcode for Entities, напротив, поддерживает Unity Physics и Havok Physics с возможностью детерминированной симуляции. Благодаря этому пакет обеспечивает полноценное прогнозирование на стороне клиента.

Прогнозирование на стороне клиента в Netcode for Entities

Netcode for Entities предоставляет развитые средства клиентского прогнозирования и компенсации задержки. Для каждой сущности на клиенте и сервере выполняется один и тот же код симуляции. Поэтому клиент может прогнозировать состояние игры по вводу игрока и давать немедленный отклик, не ожидая ответа сервера.

Получив от сервера последний снимок, клиент обновляет им все прогнозируемые сущности. Затем он запускает PredictedSimulationSystemGroup. Эта группа систем выполняет симуляцию от самого старого сохраненного тика до текущего целевого времени: откатывает и заново просчитывает состояние игры. Такой процесс исправляет расхождения и поддерживает синхронизацию с авторитетным состоянием сервера.

На сервере цикл прогнозирования выполняется один раз за кадр. Сервер обновляет авторитетное состояние игры и отправляет его клиенту.

Клиент хранит историю ввода и прогнозируемых состояний. При рассинхронизации он откатывается к последнему корректному состоянию сервера и повторно просчитывает игру с правильными входными данными. Так клиент остается синхронизированным с эталонным состоянием сервера.

Netcode for Entities также поддерживает расширенные возможности физики:

- Несколько физических миров: локальные физические симуляции, которые не нужно реплицировать по сети.

- Пользовательские физические прокси: позволяют ghost-сущностям взаимодействовать с физическими объектами, существующими только на клиенте, например с обломками.

- Детерминированная физическая симуляция: при одинаковых входных данных клиент и сервер получают одинаковый результат, поэтому их состояния остаются согласованными.

GhostPredictionSmoothingSystem сглаживает ошибки прогнозирования при переходе между прогнозируемым и авторитетным состояниями и поддерживает пользовательские алгоритмы сглаживания.

Сочетание прогнозирования, отката, компенсации задержки и сглаживания сводит к минимуму влияние сети, помогая сохранить целостность и отзывчивость игры.

Сравним подходы Netcode for GameObjects и Netcode for Entities к клиентскому прогнозированию:

Характеристика	Netcode for GameObjects	Netcode for Entities
Прогнозирование	Ожидание на клиенте: клиент предугадывает ответ сервера.	Полное прогнозирование: клиент выполняет тот же код симуляции, что и сервер.
Снижение задержки	Предугадывает результат сервера и сглаживает переходы.	Использует откат и повторную симуляцию для исправления рассинхронизации.
Снимки	Явно не используются.	Представляют состояние игры в конкретные моменты.
Компенсация задержки	Упрощённая модель с параметрами StaleDataHandling.	Полная компенсация задержки с обращением к мирам столкновений.
Физические взаимодействия	Ограничено клиентским прогнозированием с упреждением преобразований.	Взаимодействие прогнозируемых и только клиентских физических миров.
Сглаживание	Метод Smooth для ожидаемых значений.	GhostPredictionSmoothingSystem сглаживает переходы между прогнозируемым и авторитетным состоянием.
Применение	Простые игры с мгновенной визуальной обратной связью.	Сложные игры, требующие точной синхронизации состояния.

И Netcode for GameObjects, и Netcode for Entities предоставляют механизмы клиентского прогнозирования и компенсации сетевой задержки. Netcode for GameObjects предлагает упрощенную модель клиентского упреждения, а Netcode for Entities - более развитую систему с полноценным прогнозированием, откатом и компенсацией задержки.

Термины Netcode for Entities

Netcode for Entities использует Entity Component System (ECS) и Data-Oriented Technology Stack (DOTS). Этот подход отличается от рабочего процесса MonoBehaviour в Netcode for GameObjects. Ниже перечислены термины, которые могут быть вам незнакомы:

Entity (сущность): в ECS это базовая единица данных, представляющая отдельный игровой объект или компонент. Сущности легковесны и содержат только данные, но не поведение.

Game world (игровой мир): вся среда, в которой происходит игра. В нее входят все сущности, их состояния и правила взаимодействия.

Collision world (мир столкновений): состояние всех физических объектов игрового мира, включая их положение, скорость и взаимодействия. Используется для обнаружения столкновений и физической симуляции.

Snapshot (снимок, или ghost snapshot): набор данных о состоянии игры в определенный момент времени. Клиент и сервер периодически синхронизируют состояние игры с помощью снимков.

Ghost: сетевая сущность, реплицируемая между сервером и клиентами. В каждом кадре сервер отправляет клиенту снимок текущего состояния всех ghost-сущностей. Они синхронизируют сущности, чтобы все игроки видели согласованное состояние игрового мира.

Predicted ghost (прогнозируемая ghost-сущность): клиентская сущность, которую локально симулируют для немедленного визуального отклика на действия игрока и снижения воспринимаемой задержки. Используйте такие сущности для объектов под непосредственным управлением игрока и других интерактивных объектов, требующих мгновенного отклика.

Interpolated ghost (интерполированная ghost-сущность): представление серверной сущности на клиенте. Клиент отображает ее состояние по полученным от сервера снимкам, плавно смешивая их для уменьшения вызванного задержкой джиттера. Такой вариант подходит для объектов, которыми игрок не управляет напрямую и которым не нужен немедленный отклик: персонажей других игроков, NPC и управляемых сервером сущностей.

Тестирование и отладка сетевых игр

Тестирование и отладка многопользовательских игр отличаются от работы с однопользовательскими проектами. Учитывайте основные этапы рабочего процесса сетевого проекта:

- При локальном тестировании запускают несколько экземпляров игры и проверяют взаимодействие игроков. Для разработки приложения используйте сборки Player, пакет Multiplayer Play Mode и Network Scene Visualization.

- Имитируйте сетевые условия с помощью скриптов или компонента NetcodeTransport. Так можно воспроизвести задержку, джиттер и потерю пакетов, характерные для реальных сетей.

- Управление клиентскими подключениями охватывает присоединение, повторное подключение и отключение клиентов.

- Для журналирования и поиска причин неполадок используйте инструменты отладки.

- Вспомогательный инструмент командной строки автоматизирует тестирование: запускает экземпляры игры с заданными ролями и сетевыми условиями.

Локальное тестирование

Локальное тестирование - важная часть разработки многопользовательской игры: несколько экземпляров приложения имитируют взаимодействие разных игроков в сетевых условиях.

Сборки Player

Запуская несколько экземпляров исполняемого файла игры, можно создавать сеансы и присоединяться к ним. Экземпляры также можно запускать одновременно с Unity Editor, чтобы на одном устройстве имитировать хоста и подключающихся игроков.

Multiplayer Play Mode (MPPM)

Входящий в Unity 6 пакет Multiplayer Play Mode (MPPM) позволяет имитировать до четырех игроков на одном компьютере разработчика, используя одни и те же исходные ресурсы. Отдельные сборки Player не требуются, поэтому тестовые итерации занимают меньше времени.

Пользователям macOS

В macOS для запуска нескольких экземпляров приложения нужна командная строка. Выполните в Terminal команду open, чтобы запустить отдельный экземпляр приложения.

Например, команда open -n YourAppName.app в Terminal запускает отдельный экземпляр приложения YourAppName.

Имитация сетевых условий

При локальном тестировании все экземпляры игры используют один сетевой интерфейс. Задержка между клиентами практически отсутствует, поэтому для имитации реальной среды ее необходимо добавить искусственно.

Задержка, джиттер и потеря пакетов влияют на игровой процесс. Проверка приложения в неблагоприятных сетевых условиях помогает убедиться, что итоговая сборка будет надежно работать через интернет.

Набор тестовых условий зависит от целевой платформы, региона и особенностей игры. Для начала используйте задержку около 100-150 мс на настольных системах и 200-300 мс на мобильных устройствах, а также потерю 5-10% пакетов. Обязательно проверяйте джиттер и потерю пакетов: они воспроизводят реальную нестабильность сети. Дополнительные рекомендации приведены на этой странице документации.

Для локального тестирования в редакторе можно совместно использовать Network Simulator из пакета Multiplayer Tools и Multiplayer Play Mode.

Для тестирования отладочных сборок рекомендуем добавлять неблагоприятные сетевые условия с помощью Network Simulator и собственного кода: окно Network Simulator работает только в редакторе.

Для тестирования релизных сборок в Windows рекомендуем clumsy, а в macOS и iOS - Network Link Conditioner. В macOS скриптуемой альтернативой Network Link Conditioner служит dummynet: этот гибко настраиваемый инструмент входит в состав операционной системы.

Подробности см. в разделе «Системные инструменты эмуляции сетевых условий».

Тестирование клиентских подключений

Тестирование управления клиентскими подключениями помогает избежать ошибок и обеспечить плавный игровой процесс. Проверьте следующие сценарии:

Подключение клиентов:

- Присоединение к новому сеансу, повторное подключение после выхода или завершения хостинга, позднее присоединение к уже идущей игре и обработка отклоненного подключения.

- Влияет ли предыдущее состояние клиента на подключение и правильно ли сервер реплицирует состояние игры.

- Корректно ли сервер обрабатывает повторное и позднее подключение.

Отключение клиентов:

- Штатное завершение работы клиента, тайм-ауты и последствия потери соединения с хостом или сервером.

- Правильно ли сбрасываются неуничтоженные объекты, связанные с сеансом, и может ли клиент подключиться к новой игре.

Запуск сеанса хостом или сервером:

- Запуск нового сеанса после завершения предыдущего, особенно в играх с клиентским хостингом.

- Влияет ли состояние приложения перед новым сеансом на игру.

Завершение работы хоста или сервера:

- Штатное завершение работы хоста или сервера, особенно при использовании внешних сервисов, например Unity Gaming Services или Lobby.

- Оповещаются ли о завершении работы клиенты и внешние сервисы.

Тщательная проверка этих сценариев помогает обеспечить стабильную и комфортную сетевую игру.

Подробнее см. в материале «Тестирование управления клиентскими подключениями».

Приемы отладки многопользовательских игр

При отладке многопользовательских игр действуют все общие принципы разработки игр. Однако для некоторых типичных сетевых сценариев нужны специальные приемы и подходы.

Ниже - приемы, полезные при разработке многопользовательских игр в Unity:

- Отладочная графика: линии Debug.DrawRay и Debug.DrawLine помогают показывать сетевые позиции, предполагаемые направления движения и взаимодействия объектов. Они особенно полезны при параллельном просмотре записей игрового процесса нескольких участников.

- Line Renderer с поддержкой Netcode: визуальная обратная связь может показывать направление, значение или другую полезную для проекта отладочную метрику. Реализация приведена в примере скрипта Line Renderer с поддержкой Netcode.

- Текстовые журналы: позволяют отслеживать невизуальные события, например RPC, и сопутствующие сведения. Добавляйте к сообщениям сетевой тик и идентификатор клиента, чтобы по журналам было проще восстановить хронологию.

- Эмуляция сетевых условий: Network Simulator имитирует на уровне приложения искусственную задержку, джиттер и потерю пакетов, помогая находить связанные с ними ошибки. Подробнее см. в разделе «Системные инструменты эмуляции сетевых условий».

- Запись экрана: одновременно записывайте экземпляры клиента и сервера для сравнения игрового процесса в реальном времени. В отладочной сборке отмечайте каждый кадр идентификатором клиента и номером кадра - это дает ориентир при параллельном просмотре.

- Увеличение фиксированного временного шага: даже с качественной отладочной графикой и журналами бывает трудно понять происходящее при покадровом анализе. Увеличение значения Fixed Timestep, например до 0,2, делает поведение игры в каждом кадре нагляднее.

- Точки останова: при долгой паузе в отладчике сетевое соединение может завершиться по тайм-ауту. Временно увеличьте тайм-аут, чтобы избежать отключения во время остановки игры.

Дополнительные советы приведены в руководстве «Приемы и хитрости отладки многопользовательских игр».

Вспомогательный инструмент командной строки

Постоянный запуск и тестирование многопользовательских сборок из Unity Editor занимает много времени. Инструмент командной строки может автоматизировать запуск и проверку сборок вне редактора.

Начать можно с примера скрипта NetworkCommandLine. Добавьте компонент NetworkCommandLine к GameObject, чтобы скрипт вошел в сборку и был доступен через командную строку.

В окне Player Settings откройте Settings for PC, Mac & Linux Standalone > Resolution and Presentation и установите для Resolution значение Windowed. Так несколько экземпляров игры будет удобнее тестировать рядом друг с другом.

Скрипт NetworkCommandLine должен сначала проверить, что игра запущена вне редактора. Затем он считывает аргументы командной строки, определяет нужный режим (server, host или client) и запускает соответствующие службы. Это позволяет задавать сетевую роль сборки при запуске из командной строки.

Создайте сборку через File > Build Settings и протестируйте её из командной строки.

В Windows:

Откройте командную строку и перейдите в каталог со сборкой. Запускайте экземпляры сервера и клиента отдельными командами, при необходимости изменив путь. Например:

```
<Путь к проекту>\HelloWorld.exe -mode server
<Путь к проекту>\HelloWorld.exe -mode client
```

В macOS:

Откройте Terminal и используйте аналогичные команды, указав пути в формате macOS. Например:

```
<Путь к проекту>/HelloWorld.app/Contents/MacOS/<Имя проекта> -mode server
<Путь к проекту>/HelloWorld.app/Contents/MacOS/<Имя проекта> -mode client
```

При необходимости направьте вывод экземпляров игры в текстовые файлы с помощью флага -logfile: так его удобнее отслеживать и анализировать.

Multiplayer Services

Unity предоставляет ряд сервисов, упрощающих разработку сетевых многопользовательских игр. Начните с документации по пакету Multiplayer Services: он упрощает управление зависимостями между многопользовательскими сервисами. С его помощью можно:

- Быстро добавлять в игру многопользовательские возможности на базе Unity Gaming Services: Lobby, Relay, Distributed Authority, Matchmaker и Multiplay Hosting.

- Использовать новую систему сеансов как единый облачный уровень для многопользовательского игрового цикла: объединять игроков и управлять общим состоянием сеанса и участников.

- Создавать и администрировать сетевые сеансы P2P, Dedicated Game Server и Distributed Authority. Игроки могут подключаться через Matchmaker, по коду присоединения или из списка активных сеансов.

Кратко рассмотрим каждый сервис.

Matchmaker

Подбор игроков соединяет участников друг с другом или с игровыми сеансами по заданным критериям, например уровню навыка или региону, чтобы матчи были сбалансированными и увлекательными.

Интеграция сервиса Unity Matchmaker включает несколько этапов:

- Задайте критерии подбора: уровень навыка, географическую близость, задержку, предпочтения игроков и другие параметры.

- Ведите профили и рейтинги игроков с нужными для подбора параметрами. Рейтинговая система, например Elo, помогает оценивать уровень навыка и формировать сбалансированные матчи.

- Обрабатывайте запросы на подбор: сервис ищет подходящих соперников или союзников в пуле доступных игроков по заданным критериям и выбирает наилучшее сочетание.

- Назначайте найденных игроков в игровой сеанс и проверяйте, что все подключены и готовы к началу игры.

- Динамически ослабляйте критерии, если точное соответствие найти не удается, чтобы сформировать матч вовремя и не слишком снизить его качество.

Учитывайте эти факторы, чтобы Unity Matchmaker формировал честные и конкурентные матчи.

Lobby

Лобби - это предыгровая область, где игроки собираются, настраивают параметры и готовятся к началу матча. Сервис Lobby помогает участникам находить друг друга и подключаться в различных многопользовательских сценариях. Например:

- Просматривать доступные игровые сеансы, выбирать нужный и присоединяться
- Передавать другу код присоединения, чтобы он мог подключиться к вашему сеансу
- Использовать Quick Join, чтобы найти любой доступный матч и сразу войти в него
- Создавать общедоступное или закрытое лобби и приглашать друзей из внутриигрового списка
- Размещать лобби на игровом сервере, управляя доступом к серверному сеансу
- Искать лобби по заданным требованиям, например режиму игры или типу карты

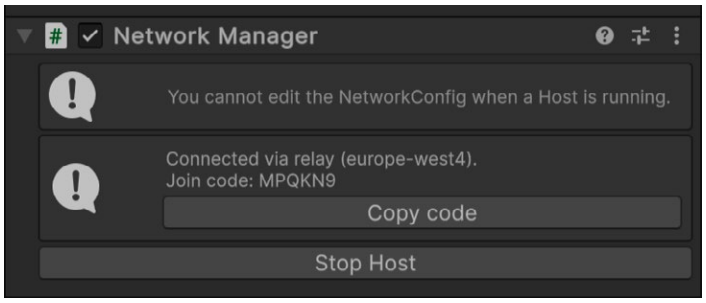
Пример Game Lobby показывает, как с помощью пакетов Lobby и Relay реализовать типичный игровой зал ожидания, включая голосовой чат Vivox. Игрок может создать лобби, а другие - найти его в общедоступном списке или подключиться по коду. Затем участники соединяются через Relay и используют Unity Transport для базовой связи в реальном времени.

Relay

Unity Relay упрощает подключение нескольких игроков с помощью кодов присоединения. Создав сеанс, хост получает уникальный код и передает его друзьям или товарищам по команде. Участники используют код для подключения. При этом конфиденциальные данные, например IP-адреса, остаются скрытыми.

В многопользовательских P2P-играх подключение клиентов осложняют NAT (Network Address Translation) и межсетевые экраны. Прямое соединение часто требует сложной настройки сети и раскрывает IP-адреса игроков, создавая риски для безопасности и конфиденциальности.

Relay решает эти проблемы, выступая промежуточным сервером и обеспечивая безопасное простое соединение клиентов. Выделенный сервер не нужен, а объем разработки сокращается. Достаточно создать код присоединения, и игроки смогут сразу подключиться.



После создания подключения Relay выдает код присоединения.

Подробнее см. в руководстве по началу работы с Unity Relay.

Multiplay Hosting

Multiplay Hosting берет на себя сложности масштабного запуска и эксплуатации инфраструктуры, позволяя команде сосредоточиться на игровом опыте. Сервис также позволяет:

- Отслеживать состояние серверов и другие аналитические данные

- Обновлять серверы без простоя

- Размещать игроков на серверах с наилучшим качеством обслуживания по данным QoS

- Контейнеризировать сборки с помощью Docker и реестра контейнеров Multiplay Hosting

Vivox

Vivox улучшает совместную и соревновательную игру благодаря внутриигровому голосовому и текстовому чату. Это управляемое облачное решение упрощает интеграцию общения и предоставляет средства доступности и соблюдения нормативных требований: преобразование речи в текст, фильтрацию чата и другие возможности.

Vivox обеспечивает общение игроков на разных платформах независимо от того, создана игра в Unreal, Unity или на собственном движке.

Подключите Vivox к игре и настройте проект в Unity Dashboard. Сервис поддерживает неограниченное число пользователей в 2D- и 3D-каналах, регулировку громкости, отключение звука и управление каналами.

В этой сессии GDC показано, как несколько многопользовательских сервисов были интегрированы в демоверсию Megacity Metro.

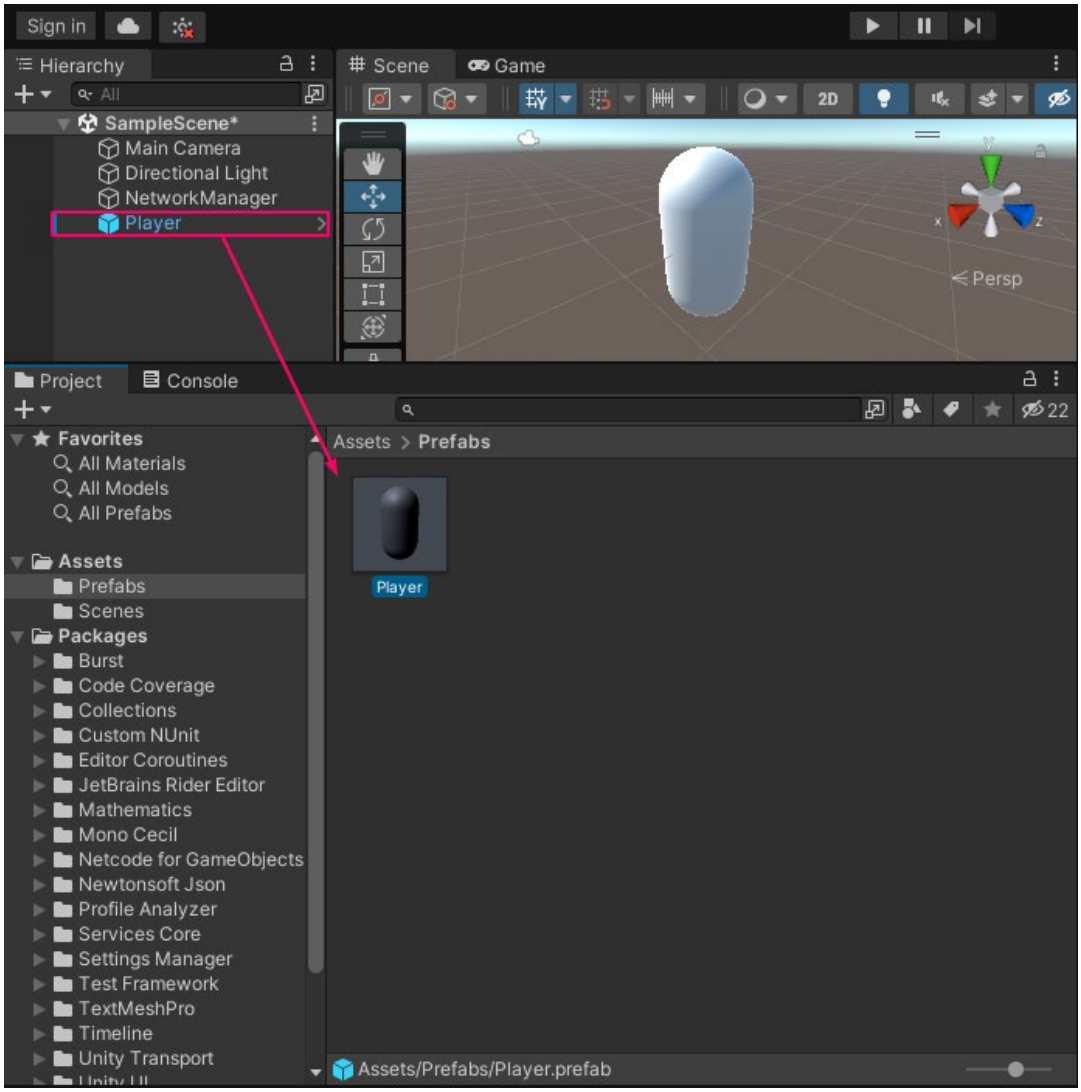
Примеры проектов и ресурсы

Unity предлагает примеры проектов, которые помогают освоить Netcode for GameObjects и Netcode for Entities. В них на практике показано, как реализовать сетевое взаимодействие в собственных многопользовательских приложениях.

Ресурсы по Netcode for GameObjects

Unity Learn: начало работы с Netcode for GameObjects

Этот курс Unity Learn пошагово объясняет, как создать и протестировать простую многопользовательскую игру в Unity, использовать и проверять удаленные вызовы процедур (RPC) и NetworkVariable. Вы также создадите простой интерфейс для режимов Host, Client и Server и добавите базовое управление перемещением в каждом режиме.

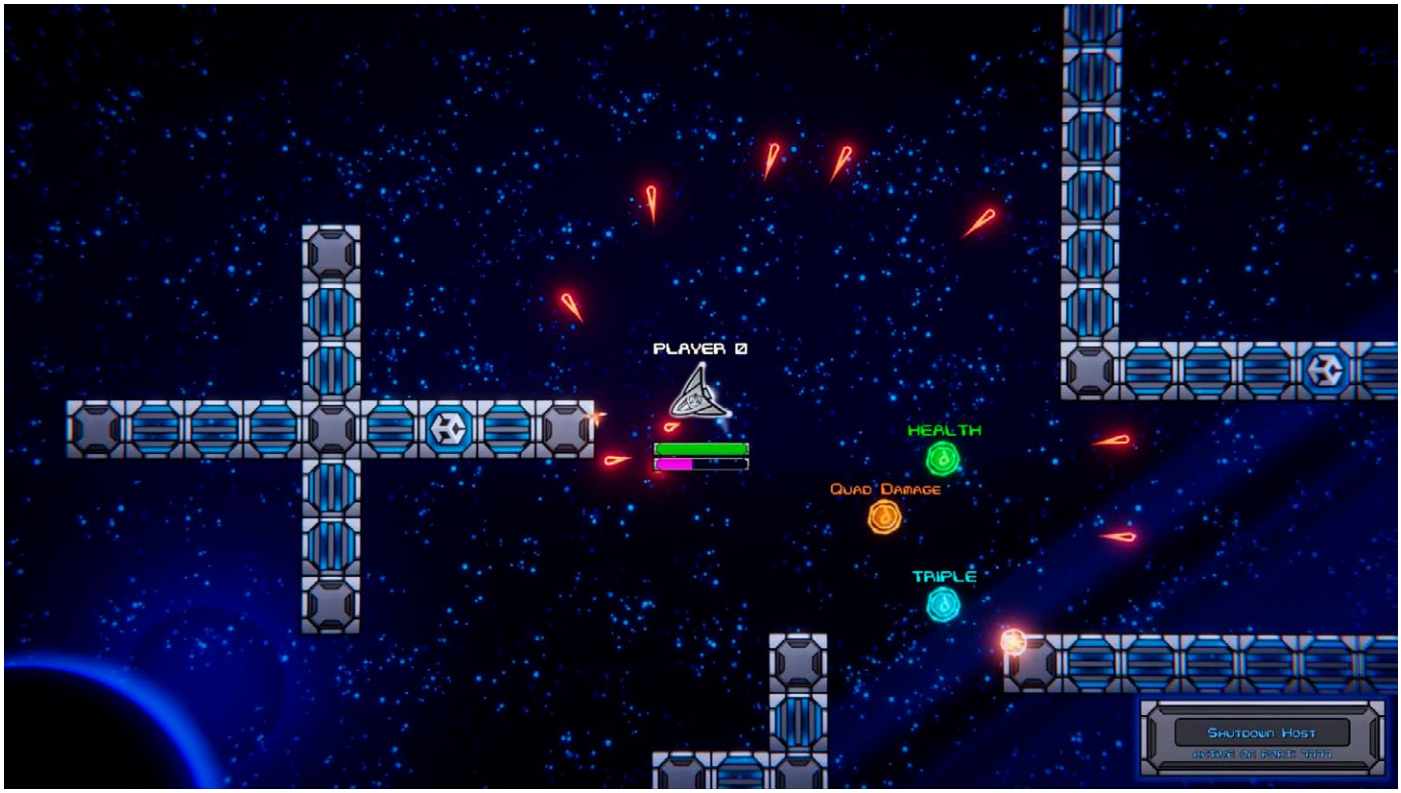


Снимок экрана из курса Unity Learn по Netcode for GameObjects

Bitesize Samples

Репозиторий Bitesize Samples содержит набор модульных примеров кода, которые можно применять в своих играх и использовать для изучения Netcode for GameObjects. В репозиторий входят:

- 2D Space Shooter Sample: физическое перемещение, эффекты состояния, NetworkVariable и пулы объектов.



2D Space Shooter из набора Bitesize Samples

- Distributed Authority Social Hub: настройка топологии, в которой управление состоянием игры распределено между несколькими клиентами, что снижает нагрузку на сервер.

- Multiplayer Use Cases Overview: типичные действия в многопользовательской среде, которые следует учитывать при проектировании игровых возможностей.

- Client Driven Sample: клиентское управление перемещением, сетевая физика, создаваемые и статически размещенные объекты, а также смена родительского объекта.

- Dynamic Addressables Network Prefabs: динамическая система префабов, позволяющая добавлять во время выполнения новые префабы, доступные для создания.

Boss Room

Boss Room - пример казуальной трехмерной кооперативной игры на базе Netcode for GameObjects. Проект помогает изучить основные концепции и шаблоны многопользовательского игрового цикла.

Boss Room представляет собой законченный фрагмент полнофункциональной кооперативной игры.

Boss Room - одновременно рабочий пример и учебный проект для разработчиков сетевых многопользовательских игр. Это самый зрелый готовый к выпуску пример Unity на базе рабочих процессов Netcode for GameObjects. Он содержит код промышленного уровня и интегрирован с сервисами Lobby и Relay.

Boss Room демонстрирует сетевые игровые механики: способности персонажей и особые атаки, сетевую физику и отслеживание состояния разрушаемых объектов, переключателей и дверей. Эти приемы показывают, как скрыть задержку и сохранить плавность игры даже в неблагоприятных сетевых условиях. Рассмотрим проект подробнее.

Игровой цикл и управление состоянием: Boss Room содержит практическую реализацию состояния как в лобби, так и во время игры. Проект показывает обработку подключений, загрузку и выгрузку сетевых сцен и корректное отключение игроков.

Интеграция с UGS: в Boss Room используются Relay, Lobby, Authentication и другие возможности UGS.

Вспомогательные скрипты и инструменты: репозиторий содержит решения, которые можно адаптировать для других проектов, включая примеры клиентской авторитетности, утилиты управления сценами и сетевые пулы объектов.

Клиент-серверная модель: в Boss Room один игрок выступает хостом (сервером), а остальные - клиентами. Централизация игровой логики снижает риск мошенничества и обеспечивает согласованное состояние игры у всех клиентов.

Стартовое меню примера Boss Room

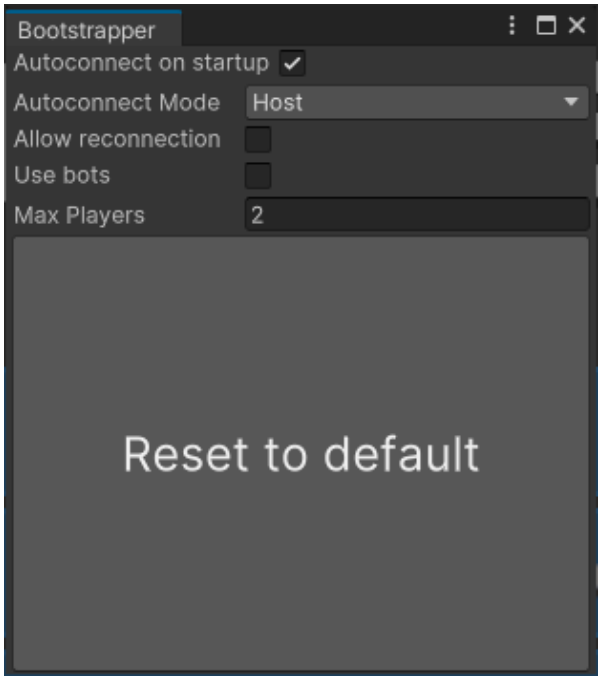
Для подробного знакомства с проектом прочитайте публикацию в блоге и посмотрите четырехчастный вебинар на YouTube «Создание готовой к выпуску многопользовательской игры с Netcode for GameObjects». В серии рассматриваются базовые игровые механики и серверная авторитетность, реализация сетевого игрового процесса и создание объектов. Также показано, как повысить устойчивость игры к действиям игроков и эффективнее использовать пропускную способность. Материалы дают практические рекомендации на примере полноценного многопользовательского проекта.

Шаблон Small Scale Competitive Multiplayer

Доступный в Unity Hub шаблон служит отправной точкой для создания и выпуска многопользовательского проекта с Netcode for GameObjects и UGS.

В шаблон входит инструмент Bootstrapper для тестирования различных сетевых режимов (Host, Client и Server) и динамических конфигураций.

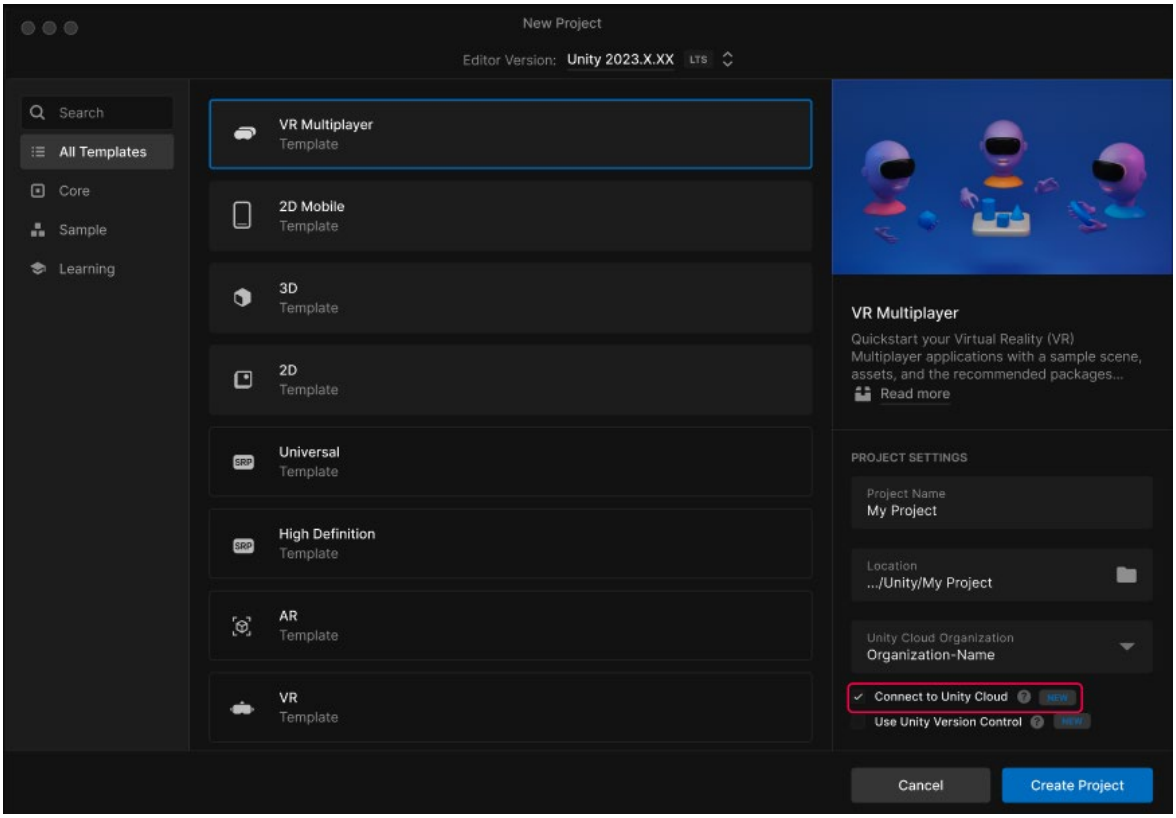
Чтобы начать, откройте проект и выполните инструкции в приветственном окне. Пройдите встроенные руководства или исследуйте проект самостоятельно, загрузив стартовую сцену через меню Multiplayer > Bootstrapper. Настройте поля Bootstrapper в соответствии с тестовым сценарием и перейдите в режим Play.



Настройте параметры Bootstrapper.

Учебные материалы и полезные ресурсы доступны в любое время через меню Tutorials > Show Tutorials.

Шаблон VR Multiplayer



Откройте шаблон VR Multiplayer в Unity Hub.

Шаблон VR Multiplayer предназначен для новых проектов под устройства OpenXR. Он содержит все необходимое для сетевых взаимодействий, голосового чата, лобби и других функций. Шаблон использует Unity Gaming Services, Netcode for GameObjects и XR Interaction Toolkit и оптимизирован для Meta Quest и других совместимых с OpenXR устройств. Основные возможности:

- Сетевые взаимодействия на базе XRI Interaction Toolkit и Netcode for GameObjects

- Голосовая связь через Vivox

- Подключение игроков из любой точки мира с помощью Relay и Lobby

- Сетевые аватары и руки

- Кроссплатформенная работа через OpenXR

Загрузите шаблон Unity VR Multiplayer для Unity 2022 LTS и Unity 6 Preview через Unity Hub и ознакомьтесь с кратким руководством.

Ресурсы по Netcode for Entities

Начало работы с Netcode for Entities

Этот вебинар по запросу подробно рассматривает Megacity Metro - крупномасштабный кроссплатформенный пример многопользовательской игры Unity, созданный с Netcode for Entities и Unity Gaming Services. Материал рассчитан на пользователей, уже знакомых с DOTS и готовых создавать технически сложные многопользовательские игры.

Примеры ECS Netcode

Примеры демонстрируют множество базовых и расширенных возможностей: синхронизацию, сценарии подключения, интеграцию с Unity Physics и многое другое. Начните с руководства Networked Cube, где рассматриваются:

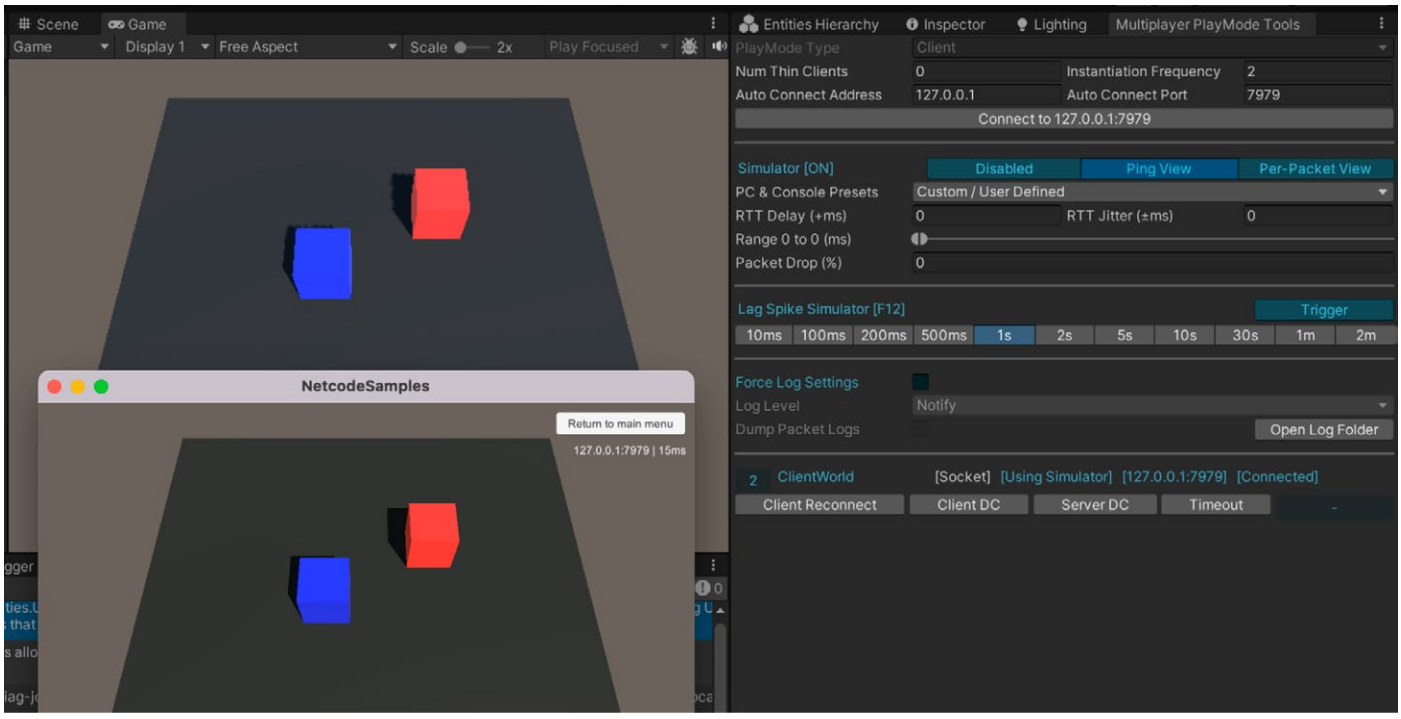
- Установление соединения с сервером

- Обмен данными с сервером

- Создание синхронизированных сущностей на сервере

- Создание автономных сборок сервера и клиента

- Запуск сервера и клиента в режиме Play внутри редактора



Руководство Networked Cube: запуск в редакторе и в автономной сборке

ECS Network Racing

Этот пример многопользовательской гоночной игры демонстрирует рекомендуемые приемы работы с Unity Netcode for Entities. В нем реализована клиент-серверная архитектура с прогнозированием на стороне клиента, интерполяцией и компенсацией задержки.

ECS Racing демонстрирует расширенные многопользовательские возможности.

Megacity Metro

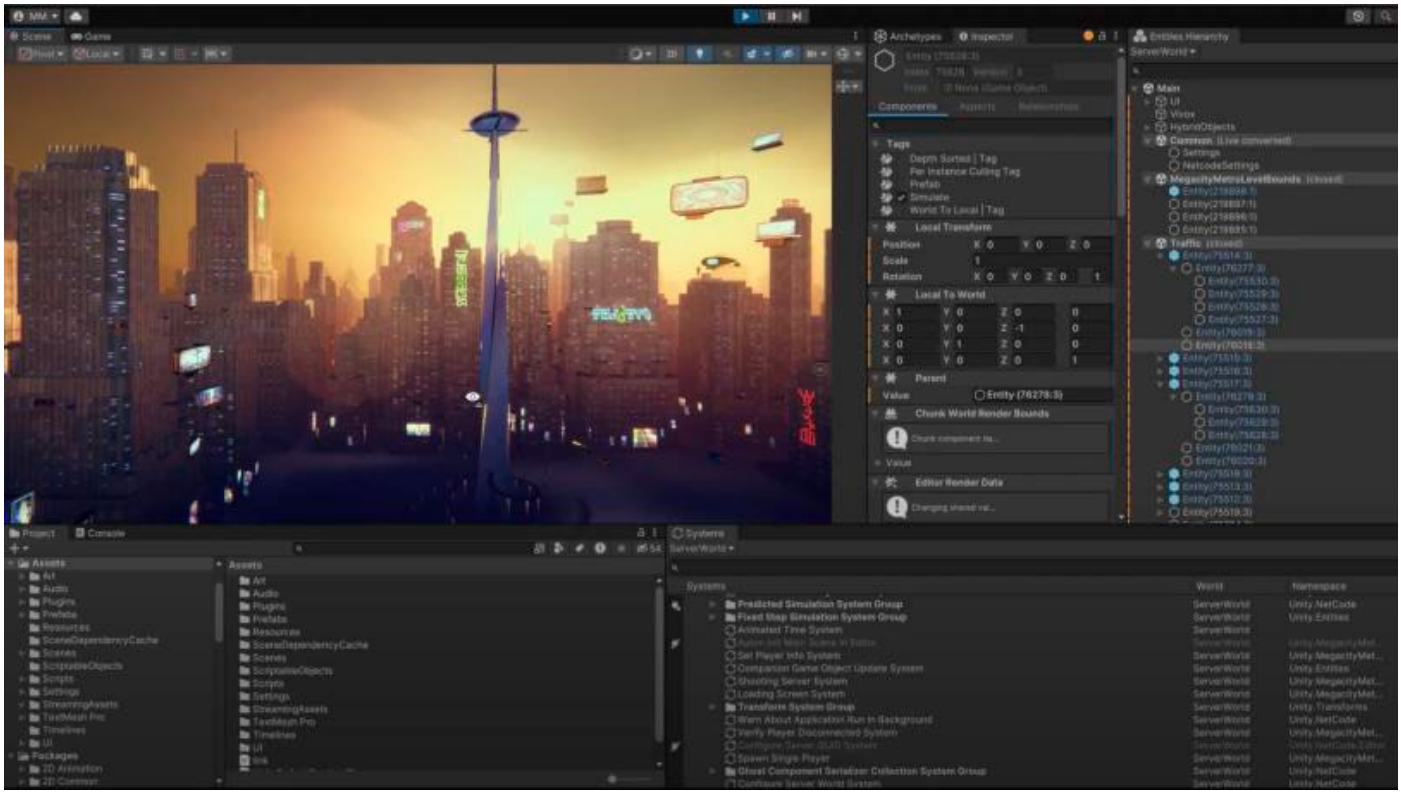
Megacity Metro для Unity 6 и Unity 2022 LTS - масштабируемая кроссплатформенная демонстрация новейших технологий Unity, включая пакет Netcode for Entities. Этот многопользовательский экшен от третьего лица поддерживает более 128 игроков одновременно.

Пример подойдет тем, кто хочет освоить игровой процесс с авторитетным сервером и применение UGS на всех этапах создания многопользовательской игры.

Демоверсия Unity Megacity Metro поддерживает более 128 игроков.

Узнайте больше о создании масштабных игр с ECS for Unity и многопользовательскими решениями Unity. ECS дает опытным разработчикам дополнительный контроль и детерминизм, необходимые для более амбициозных проектов.

Megacity Metro демонстрирует интерполяцию, прогнозирование на стороне клиента и компенсацию задержки. Загрузите проект, чтобы изучить Multiplay Hosting, Matchmaker, Vivox Voice Chat и другие сервисы.



Megacity Metro демонстрирует применение Netcode for Entities в многопользовательском экшене.

Экспериментальный пакет Multiplayer Services

Для многопользовательской игры необходимо объединить несколько продуктов и сервисов. Новый пакет Multiplayer Services (com.unity.services.multiplayer) упрощает их интеграцию, управление зависимостями и взаимодействие с продуктами.

Multiplayer Services - единое решение для добавления сетевого многопользовательского режима. Пакет работает на базе UGS и объединяет возможности Relay, Lobby и других сервисов в новой системе Sessions, которая упрощает соединение групп игроков.

Multiplayer Services позволяет создавать P2P-сеансы и предлагает несколько способов присоединения: по коду, через список активных сеансов или с помощью Quick Join.

Дальнейшие шаги

Теперь у вас есть прочная база сетевых концепций и практический опыт работы с Netcode for GameObjects. Продолжайте осваивать многопользовательскую разработку:

Изучите Unity Learn: новые пошаговые материалы делают освоение многопользовательской разработки еще проще. Первый курс Multiplayer на Unity Learn познакомит с основами Netcode for GameObjects и проведет вас через создание первого сетевого проекта.

Исследуйте примеры проектов: подробнее разберите Boss Room, Galactic Kittens и Bitesize Samples. Изменяя эти проекты, вы освоите новые приемы и рекомендуемые подходы к сетевой многопользовательской разработке.

Экспериментируйте с многопользовательскими сервисами: не нужно заново изобретать готовые решения. Relay, Lobby и Matchmaker можно сразу внедрять в проекты, заметно сокращая время разработки.

Освойте расширенные методы: по мере роста требований проекта обратите внимание на Netcode for Entities. Пакет поддерживает прогнозирование на стороне клиента и компенсацию задержки для максимальной сетевой производительности. Эти возможности показаны в примерах Megacity Metro и ECS Racing.

Успехов в разработке сетевых игр!


Unity®