Сетевые решения, интеграция SDK

Клиент-серверная архитектура, фреймворки, примеры сетевых игр, подключаем рекламу

Что будет

1. Что такое многопользовательская игра — примеры и требования
2. Ключевые проблемы: задержки, потеря пакетов, состояния, читерство, масштабирование
3. Архитектуры, транспортные уровни, синхронизация
4. Обзор фреймворков
5. Интеграция SDK и частые проблемы
6. Пример UnityAds

Многопользовательская игра

Многопользовательская игра (multiplayer game) — это игра, в которой несколько игроков взаимодействуют в одной игровой среде одновременно, соединяясь через сеть (LAN, интернет) и обмениваясь данными о состоянии игры в реальном времени.

Примеры на Unity

Among Us

Casual, кооператив

Peer-to-peer с relay (через Photon)

Photon Realtime / PUN 2

Простая синхронизация позиций, RPC для событий (“убийство”, “собрание”), минимальный трафик

Fall Guys

Party PvP

Client–Server (authoritative)

Собственный backend + Photon Fusion (на ранних версиях)

Сессионная игра, обработка коллизий и событий на сервере

Rust

Survival MMO

Dedicated authoritative server

Собственный C# сервер, Unity Transport (ENet)

Полностью серверная симуляция мира, синхронизация позиций, состояния, ресурсов

Escape from Tarkov

PvP Shooter

Dedicated server

Кастомный .NET backend + Unity Transport

Сервер проверяет каждый выстрел, интерполяция/репликация состояния

Valheim

Survival, co-op

Peer-hosted (один клиент = хост)

LiteNetLib + кастомный код

Детерминированная синхронизация мира, оптимизация под 10 игроков

Genshin Impact

Open World Action RPG с online-функциями

Free-to-Play, client-server

Проприетарный backend на C++ / Go / Lua + Redis + MongoDB

Unity используется только как клиентская часть, сильно кастомизированная: собственный Netcode Layer (вместо UNet, Mirror и т.д.), своя система ресинхронизации мира и state prediction

Характеристики

Несколько игроков

Два и более активных участника, каждый управляет своим персонажем или элементом игры

Характеристики

Сетевая коммуникация

Игровые данные (позиции, действия, события) передаются между клиентами и сервером через интернет

Характеристики

Синхронизация

У всех участников игра должна отображаться согласованно — одинаковое состояние мира при сетевых задержках

Характеристики

Архитектура

Обычно используется модель клиент–сервер (server authoritative) или peer-to-peer

Характеристики

Безопасность и честность

Сервер контролирует состояние, предотвращает читы, верифицирует команды

Характеристики

Коммуникация между игроками

Чат, голос, обмен предметами, взаимодействие в одной игровой сессии

Классификация

Локальная (LAN / Split-screen)

Mortal Kombat, FIFA (локально)

Без интернета, внутри одной сети или устройства

Классификация

Кооперативная (Co-op)

Left 4 Dead, It Takes Two

Игроки вместе против ИИ, важна синхронизация состояния

Классификация

Сессионная PvP / PvE

Counter-Strike, Dota 2

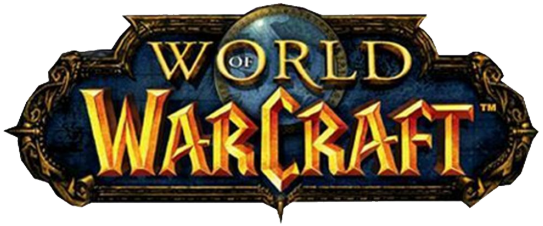
Быстрые матчи, матчмейкинг, серверные сессии

Классификация

Массовая онлайн (ММО)

World of Warcraft, GTA Online

Сотни/тысячи игроков, распределённые серверы, масштабирование



Классификация

Асинхронная (по очереди)

NFS, Words With Friends, шахматы онлайн

Игроки действуют с задержкой, обмен данными через API/бэкенд

Требования

Надёжная архитектура

- Разделение логики клиента и сервера
- Сервер — источник истины (authoritative server)
- Масштабирование и устойчивость к нагрузкам

Требования

Минимальная задержка (latency)

- Использование UDP или специализированных транспортов
- Интерполяция и предсказание действий на клиенте

Требования

Согласованность состояния

- У всех игроков объекты должны быть в одинаковом положении
- Синхронизация позиций, здоровья, событий

Требования

Безопасность и античит

- Проверка входных данных на сервере
- Запрет прямого управления объектами со стороны клиента

Требования

Управление подключениями

- Подключение, отключение, восстановление соединения
- Matchmaking (поиск игр), Lobby, управление комнатами

Требования

Обновления и совместимость

- Поддержка разных версий клиента и сервера
- Возможность патчей без разрыва сессий

Требования

Тестирование и отладка

- Проверка при высокой задержке, packet loss
- Логирование, профилирование, нагрузочные тесты

Типовые архитектуры

Async моб. игра (Firebase / Nakama)

Архитектура: REST/WebSocket backend

Основные требования: авторизация, сохранение состояния, синхронные события

Типовые архитектуры

2 игрока в коопе (Unity + Mirror)

Архитектура: Client–Server, хост = сервер

Основные требования: простая синхронизация, низкий пинг

Типовые архитектуры

PvP шутер 10×10 (Unity + Photon Fusion)

Архитектура: Dedicated server

Основные требования: авторитет сервера, 60–120 updates/sec, предсказание

Типовые архитектуры

MMO 500+ игроков (Nakama / SmartFox)

Архитектура: Шардинг серверов, persistent world

Основные требования: масштабируемость, хранилище состояния, безопасность

Peer-to-Peer (P2P)

Все клиенты напрямую обмениваются данными друг с другом.

Плюсы

- Нет выделенного сервера
- Минимальные затраты на инфраструктуру
- Простая реализация в Unity (Mirror / Photon PUN)

Минусы

- Высокий риск читов (клиент “владеет” своими данными)
- Проблемы с NAT / firewall
- Сложная синхронизация при большом числе игроков

Peer-to-Peer (P2P)

Among Us (Photon PUN 2)

- Один клиент создаёт комнату, остальные подключаются напрямую (через Photon relay)
- Все игроки “равны”, но инициатор комнаты — условный “лидер”

Host-Client

Один из игроков запускает локальный сервер, остальные клиенты подключаются к нему.

Плюсы

- Простая реализация (Mirror, FishNet)
- Минимальная задержка для хоста
- Не требует дорогих серверов

Минусы

- Если хост выходит — сессия завершается
- Возможна задержка у остальных
- Уязвимость к читам (авторитет у хоста)

Host-Client

CS 1.6

Warcraft

Phasmophobia

Один игрок создаёт комнату, приглашает друзей, его ПК исполняет роль сервера.

Client–Server (Dedicated)

Отдельная программа-сервер управляет всей логикой игры, а клиенты — только визуализаторы.

Плюсы

- Полный контроль (сервер — источник истины)
- Защита от читов
- Масштабируемость (можно добавить матчмейкинг, балансировщики и т.д.)

Минусы

- Дорогая инфраструктура (нужны серверы)
- Более сложная архитектура (отдельный билд сервера)
- Требует оптимизации сетевого трафика

Client–Server (Dedicated)

Genshin Impact

Rust

Fortnite

PUBG

Counter-Strike 2

World of Warcraft

Client–server (dedicated) — это индустриальный стандарт для всех серьёзных онлайн-игр!

Relay / Proxy / Hybrid

Клиенты подключаются через промежуточный relay-сервер, который пересылает пакеты (но не симулирует игру).

Плюсы

- Обходит NAT/firewall — можно играть из любой сети
- Нет необходимости в выделенном сервере логики
- Хорошо подходит для мобильных и P2P игр

Минусы

- Relay = дополнительная задержка
- Relay не защищает от читов (в отличие от авторитетного сервера)

Relay / Proxy / Hybrid

Photon Realtime или **Unity Relay Service (UGS)**

Клиенты подключаются через relay, который просто пересылает пакеты между участниками.

Асинхронная архитектура

Игроки не взаимодействуют одновременно, но обмениваются результатами через backend (REST API).

Плюсы

- Не требует постоянного соединения
- Дёшево масштабируется
- Отлично подходит для мобильных и казуальных игр

Минусы

- Нет real-time взаимодействия
- Менее “живой” опыт

Асинхронная архитектура

NFS: No Limits

Clash Royale

Wordle / Chess.com

Ключевые проблемы. Задержка

Latency (пинг) — это время, за которое пакет данных проходит от клиента до сервера и обратно. Обычно измеряется в миллисекундах (ms).

Пример:

Игрок нажал "стрелять"
→ пакет уходит на сервер (50ms)
→ сервер подтверждает попадание → клиент получает ответ (ещё 50ms)
→ итого задержка ≈ 100ms.

Задержка (Latency)

Почему это проблема:

- Управление становится “тормозным”
- Игрок видит результат своих действий с опозданием
- В шутерах и экшенах это делает игру нечестной

Задержка (Latency)

Какие есть решения:

- Client-side prediction — клиент “угадывает” результат (например, сразу двигает персонажа) и потом сверяется с сервером
- Lag compensation — сервер “перемещает” время назад при обработке попадания, чтобы учесть пинг стрелка
- Interpolation / Extrapolation — клиент плавно интерполирует между полученными состояниями других игроков
- Региональные сервера — распределение по регионам для снижения RTT

RTT (Round Trip Time) — это время, за которое пакет данных проходит от клиента до сервера и обратно

Потеря пакетов (packet loss)

Пакеты данных, отправленные по сети (UDP), могут не дойти до адресата. Обычно теряется от 0.1% до 5% пакетов.

Почему это проблема:

- Позиции игроков “телепортируются”
- Анимации дёргаются
- В шутерах могут “пропадать” выстрелы

Потеря пакетов (packet loss)

Какие есть решения:

- Reliable UDP (ack-based) — клиент подтверждает получение пакета, сервер переотправляет, если не дошёл
- Interpolation buffer — клиент отображает состояние с небольшой задержкой (например, 100ms буфер), чтобы сгладить пропуски
- Redundancy — критичные пакеты дублируются (например, позиция + ориентация)
- Snapshot delta compression — сервер регулярно посылает “слепки” состояний, клиент восстанавливает недостающие

Согласованность состояния (state synchronization)

Игровое состояние (позиции, хп, инвентарь, флаги и т.п.) должно быть одинаковым для всех клиентов. Любые рассинхроны приводят к багам и “фантомным” событиям.

Почему это проблема:

- Один игрок видит, что сундук открыт, другой — что он закрыт
- В коопе монстр “убит” у одного, но “жив” у другого
- В PvP — несогласованные попадания, “double kill”

Согласованность состояния (state synchronization)

Какие есть решения:

- Authoritative server — сервер хранит единую версию состояния и рассылает всем обновления
- Client reconciliation — клиент корректирует своё состояние по данным сервера
- State replication — сервер рассылает snapshot’ы мира (как Mirror, Photon Fusion)
- Deterministic simulation — все клиенты считают одно и то же (lockstep, используется в RTS)

Читерство (cheating)

- Client-side manipulation — изменение скорости, здоровья, выстрелов
- Packet injection — отправка ложных пакетов (“я убил врага”)
- Memory hacks — изменение данных в памяти клиента
- Aim bots / wall hacks — манипуляция логикой отображения
- Speed hacks — искажение таймингов клиента

Читерство (cheating)

Какие есть решения:

- Authoritative server logic — сервер решает всё: попадания, урон, позиции
- Anti-tamper — проверка целостности клиента (например, Easy Anti-Cheat)
- Server-side sanity checks — сервер валидирует все действия (например, не прыгай 10 раз в секунду)
- Obfuscation / encryption — шифрование сетевых сообщений
- Telemetry / behavior analysis — анализ подозрительных паттернов

Масштабирование (scalability)

Как поддерживать тысячи или миллионы игроков, не перегружая серверы?

Почему это проблема:

- Ограничения по CPU и RAM на одном инстансе
- Необходимость создавать новые игровые сессии (match instances)
- Обеспечение низкого пинга по регионам

Масштабирование (scalability)

Какие есть решения:

- Sharding (разделение миров) — каждый сервер обслуживает отдельную группу игроков
- Match-based servers — каждый матч это отдельный инстанс
- Autoscaling (облачные контейнеры) — серверы автоматически поднимаются / гасятся по нагрузке
- Microservices backend — разделение API на auth, economy, chat, telemetry
- Load balancing — балансировщик направляет новых игроков в свободные регионы

Транспортные уровни

1. TCP — Transmission Control Protocol
2. UDP — User Datagram Protocol
3. Reliable UDP — гибридный подход

TCP

TCP — это надёжный поток данных с гарантированной доставкой и порядком. Используется в HTTP, FTP, SMTP и т.д.

Как работает TCP:

1. Устанавливается соединение (handshake): клиент ↔ сервер
2. Каждому пакету присваивается номер
3. Сервер подтверждает (ACK), что пакет дошёл
4. Если пакет потерялся — пересылается заново
5. Все данные приходят в правильном порядке

TCP

Плюсы

- Гарантия доставки — данные не теряются
- Сохранение порядка — пакеты приходят в нужной последовательности
- Надёжность — хорошо подходит для важной информации (логин, чат, покупки)

Минусы для игр

- Задержка (latency) — потеря одного пакета “блокирует” весь поток до его повторной доставки (head-of-line blocking)
- Избыточность — много служебных данных, низкая эффективность
- Нет контроля над временем — TCP сам решает, когда и как переслать данные
- “Фризы” — если пакет пропал, клиент ждёт, и картинка “подвисает”

TCP

Где TCP уместен в играх:

- Авторизация / логин
- Загрузка инвентаря, профиля
- REST / HTTP-запросы (API)
- Асинхронные события (чат, статистика)

UDP

UDP — это простой и быстрый способ отправки пакетов без гарантий. Не устанавливает соединения, не ждёт подтверждений.

Как работает UDP:

- Клиент просто “кидает” пакет на сервер
- Сервер либо получает, либо нет — протокол не проверяет
- Пакеты могут прийти не по порядку, или вообще потеряться

UDP

Плюсы

- Минимальные задержки — нет подтверждений, данные сразу отправляются
- Идеально для реального времени — можно посылать 10–30 апдейтов в секунду
- Управляемость — игра сама решает, что делать при потере пакета

Минусы

- Потеря пакетов — не все данные доходят
- Нет гарантии порядка — данные приходят “вперемешку”
- Нет встроенной надёжности — всё нужно реализовывать вручную

TCP

Где UDP используется в играх:

- Позиции игроков
- Движения, повороты, события в реальном времени
- Физика и состояния объектов
- Выстрелы, столкновения, спавны

Reliable UDP

Reliable UDP — это гибрид: скорость UDP + надёжность TCP только для важных пакетов. Фреймворки для игр (Mirror, Photon, Netcode for GameObjects, ENet) добавляют надёжность поверх UDP вручную.

Как работает:

1. Основа — обычный UDP
2. Для критичных пакетов (например, “игрок умер”) добавляется: нумерация, подтверждения (ACK), переотправка при потере
3. Нерелевантные пакеты (например, “позиция игрока”) — не пересылаются, если потерялись

Reliable UDP

Быстрее TCP — нет “блокировок” потока

Гибкость — можно решать, какие пакеты важны

Управляемая надёжность — игра сама решает, когда переслать

Идеально для игр — именно так работает большинство игровых фреймворков

Фреймворки

- Mirror
- Photon
- NGO
- FishNet
- DarkRift
- LiteNetLib
- Nakama
- Colyseus
- Shardy

Mirror

Прост в освоении. Основан на старом UNet, активно развивается. Open Source.

Особенности:

- Сервер авторитарный (Dedicated или Host)
- Автоматическая синхронизация NetworkBehaviour
- Поддержка WebSockets и Telepathy

https://github.com/MirrorNetworking/Mirror

Mirror

- NetworkManager — управляет подключением, сценами, спавном игроков
- NetworkBehaviour — базовый класс для сетевых скриптов
- NetworkIdentity — уникальный ID сетевого объекта
- SyncVar — переменные, синхронизируемые между клиентом и сервером
- Command ([Command]) — метод, вызываемый с клиента, выполняется на сервере
- ClientRpc ([ClientRpc]) — метод, вызываемый на сервере, выполняется на всех клиентах
- TargetRpc — аналог ClientRpc, но только для одного клиента
- NetworkTransform / NetworkAnimator — синхронизация позиций, анимации и т.д.

Mirror

```
using Mirror;

public class Player : NetworkBehaviour
{
    [SyncVar] public int health = 100;

    [Command]
    void CmdTakeDamage(int amount)
    {
        health -= amount;
        RpcUpdateHealth(health);
    }

    [ClientRpc]
    void RpcUpdateHealth(int newHealth)
    {
        Debug.Log("Health: " + newHealth);
    }
}
```

- [Command] → клиент вызывает → выполняется на сервере
- [ClientRpc] → сервер вызывает → выполняется у всех клиентов
- [SyncVar] → автоматически синхронизируется при изменении на сервере

Mirror


```
[SyncVar(hook = nameof(OnHealthChanged))] public int health;

void OnHealthChanged(int oldValue, int newValue)
{
    Debug.Log($"Health changed from {oldValue} to {newValue}");
}
```

Если нужно реагировать на изменение переменной, Mirror сам отправит новое значение на клиент, и вызовет метод-обработчик.

Mirror

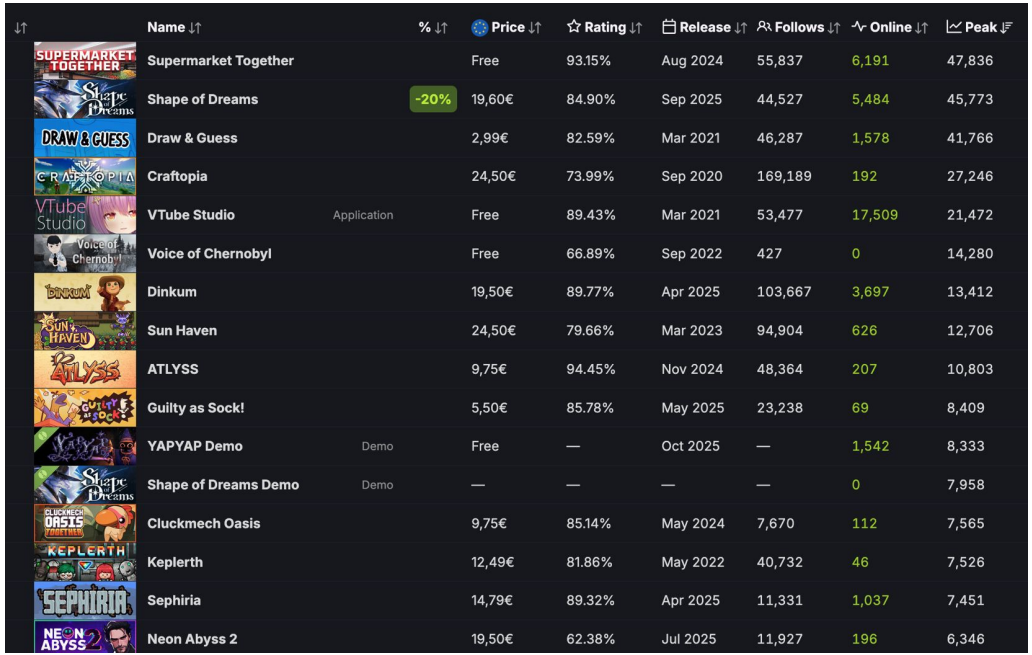
Плюсы

- Open Source, MIT
- Прост в освоении
- Активное комьюнити
- Поддерживает WebGL
- Полный контроль над логикой

Минусы

- Нужно хостить сервер
- Не поддерживает matchmaking из коробки
- Нет P2P
- Не масштабируется как Photon
- Требует DevOps для продакшена

Mirror



	Name		%	Price	Rating	Release	Follows	Online	Peak
	Supermarket Together			Free	93.15%	Aug 2024	55,837	6,191	47,836
	Shape of Dreams		-20%	19,60€	84.90%	Sep 2025	44,527	5,484	45,773
	Draw & Guess			2,99€	82.59%	Mar 2021	46,287	1,578	41,766
	Craftopia			24,50€	73.99%	Sep 2020	169,189	192	27,246
	VTube Studio	Application		Free	89.43%	Mar 2021	53,477	17,509	21,472
	Voice of Chernobyl			Free	66.89%	Sep 2022	427	0	14,280
	Dinkum			19,50€	89.77%	Apr 2025	103,667	3,697	13,412
	Sun Haven			24,50€	79.66%	Mar 2023	94,904	626	12,706
	ATLYSS			9,75€	94.45%	Nov 2024	48,364	207	10,803
	Guilty as Sock!			5,50€	85.78%	May 2025	23,238	69	8,409
	YAPYAP Demo	Demo		Free	—	Oct 2025	—	1,542	8,333
	Shape of Dreams Demo	Demo		—	—	—	—	0	7,958
	Cluckmech Oasis			9,75€	85.14%	May 2024	7,670	112	7,565
	Keplerth			12,49€	81.86%	May 2022	40,732	46	7,526
	Sephiria			14,79€	89.32%	Apr 2025	11,331	1,037	7,451
	Neon Abyss 2			19,50€	62.38%	Jul 2025	11,927	196	6,346

Почти 1500 игр на Mirror в Stream!

https://steamdb.info/tech/SDK/Mirror/?sort=rating_desc

Photon

PUN / Fusion / Quantum. SaaS-платформа (сервер у Photon). Коммерческая (до 20 CCU — бесплатно)

Преимущества:

- Не нужно писать сервер
- Поддержка room-based логики
- Интеграция в Unity Asset Store

https://www.photonengine.com

Photon

Photon — это облачный серверный бэкенд для реального времени: клиенты подключаются не друг к другу, а к дата-центру Photon Cloud.

Photon сам управляет:

- подключением,
- комнатами (rooms),
- синхронизацией событий (RPC, properties),
- авторизацией,
- репликацией состояния

Photon

Использует модель комнат (Room Based Model):

- Игроки объединяются в комнаты (matches)
- В каждой комнате может быть “Master Client” — авторитетная сторона
- Можно реализовать Dedicated Server, если нужен строгий контроль

Типы передачи данных:

- Events — одноразовые события (например, выстрел)
- Room Properties — постоянные данные комнаты (карта, очки)
- RPC / RaiseEvent — вызовы между клиентами

Photon

Photon PUN 2	High-Level	Простые проекты, казуальные игры
Photon Fusion	Modern High-Performance	Реал-тайм экшен, низкая RTT
Photon Quantum	Deterministic Engine	eSports, RTS, MOBA, физика с предсказанием
Photon Realtime SDK	Low-Level API	Собственная логика без Unity
Photon Voice / Chat	Модули	Голос, чат, friend-list и т.п.

Photon PUN 2

Photon PUN 2 (Photon Unity Networking) — старейшая и самая простая версия. Базируется на концепции комнат и владельцев объектов.

Плюсы

- Прост в освоении
- Есть готовый матчмейкинг и лобби
- Бесплатен до 20 параллельных игроков
- Работает в облаке (не нужен сервер)
- Множество примеров и документации

Минусы

- Устаревшая архитектура
- Неоптимален для шутеров и быстрых игр
- Нет детерминизма, сложно дебажить состояние
- Невозможно самостоятельно масштабировать облако (только через Photon)

Photon Fusion

Photon Fusion — современный фреймворк, созданный как замена PUN.

Поддерживает 3 режима работы:

- Shared — один игрок = хост (авторитетный)
- Server Authoritative — отдельный сервер
- Client Predicted — предсказание и reconciliation для реал-тайм

Особенности

- Ультранизкая RTT (≈ 10 мс)
- Rollback Reconciliation
- Поддерживает Host Migration
- Есть Tick-based логика (удобна для шутеров)

Photon Fusion

Плюсы

- Современная архитектура
- Высокая производительность (60 FPS)
- Поддержка Dedicated Server и Host Mode
- Встроенные Prediction / Reconciliation
- Простая интеграция в Unity через Installer

Минусы

- Бесплатный лимит тот же (20 CCU)
- Нужно понимание сетевых тиков и rollback
- Логика сложнее, чем в PUN
- Облако Photon нельзя настроить вручную

Photon Quantum

Quantum — это детерминированный мультиплеерный движок (не библиотека). Используется в AAA и eSports играх для идеальной синхронизации.

Особенности

- Полностью детерминированная физика (каждый кадр одинаков на всех клиентах)
- Работает по модели “Lockstep” (кадр не просчитывается, пока все игроки не отправят инпут)
- Поддерживает rollback и replay
- Пишется в собственном ECS-подобном API

Photon Realtime SDK

Более низкоуровневая библиотека без Unity-специфичных компонентов.

Позволяет реализовать собственный протокол, пользовательские серверные плагины.

Photon

Плюсы

- Не нужен свой сервер — всё в облаке
- Простая интеграция в Unity
- SDK с примерами и Asset Store пакетами
- Поддержка всех платформ (включая WebGL)
- API для Lobby, Voice, Chat
- Отличная документация и форум

Минусы

- Зависимость от облака
- Ограничение на CCU в бесплатном тарифе
- Приватный код — нельзя всё настроить самостоятельно
- RTT выше, если датацентр далеко
- Проблемы с веб-сокетами на WebGL (редко)

Photon

Totally Accurate Battle Simulator		18,99€	96.47%	Apr 2021	347,507	1,262	13,674
WolfQuest: Anniversary Edition	-34%	19,13€	94.88%	Jul 2025	31,075	510	2,383
Clone Drone in the Danger Zone		16,79€	94.77%	Jul 2021	33,058	287	1,944
R.E.P.O.	-35%	6,33€	94.76%	Feb 2025	262,093	59,217	271,571
Monster Prom 4: Monster Con		14,99€	94.00%	Apr 2025	9,245	10	1,839
ROUNDS		5,49€	93.30%	Apr 2021	21,549	423	2,269
Content Warning		7,49€	93.21%	Apr 2024	100,993	788	204,439
Monster Prom 2: Monster Camp		11,99€	93.16%	Oct 2020	16,363	10	2,252
Monster Prom 3: Monster Roadtrip		11,99€	93.09%	Oct 2022	13,038	13	1,774
Retrowave		2,99€	92.89%	May 2022	11,932	12	638
PlateUp!	-30%	18,29€	92.69%	Aug 2022	41,226	797	12,460
Snail Simulator		4,99€	92.68%	Nov 2023	1,191	50	317
Superliminal		19,50€	92.59%	Nov 2020	66,202	68	1,439
PEAK	-38%	4,64€	92.52%	Jun 2025	135,765	43,433	170,759
Liftoff®: FPV Drone Racing		19,99€	92.42%	Sep 2018	34,326	629	832
Descenders		22,99€	92.16%	May 2019	30,485	341	1,954

Более 4000 игр на Photon в Stream!

https://steamdb.info/tech/SDK/Photon/?sort=rating_desc

Unity Netcode for GameObjects (NGO)

Официальный фреймворк Unity. Бесплатный (MIT). Поддерживает Unity Transport (UDP, Relay).

Особенности:

- Совместим с Unity Relay & Lobby (через Unity Gaming Services)
- Прост в интеграции
- Хорошо для казуальных и кооперативных игр

https://docs-multiplayer.unity3d.com

Unity Netcode for GameObjects (NGO)

- NetworkManager — центр сетевого управления: старт/стоп хоста или клиента
- NetworkObject — компонент, который помечает GameObject как сетевой
- NetworkBehaviour — наследник MonoBehaviour с сетевой логикой (RPC, SyncVar)
- NetworkVariable — синхронизируемая переменная между клиентом и сервером
- RPC (ServerRpc / ClientRpc) — удалённые вызовы методов
- NetworkTransform / NetworkAnimator — компоненты для синхронизации позиций и анимаций

Unity Netcode for GameObjects (NGO)


```
using Unity.Netcode;
using UnityEngine;

public class PlayerController : NetworkBehaviour
{
    public NetworkVariable<Vector3> Position = new NetworkVariable<Vector3>();

    void Update()
    {
        if (IsOwner)
        {

            Vector3 move = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
            transform.position += move * Time.deltaTime * 5f;


            Position.Value = transform.position;
        }
        else
        {

            transform.position = Position.Value;
        }
    }
}
```

Unity Netcode for GameObjects (NGO)

По умолчанию:

- Unity Transport (UTP) — надстройка над UDP, с реализацией reliable/unreliable channels
- Также можно использовать custom transport adapters (например, для Relay или Steamworks).

Unity Netcode for GameObjects (NGO)

Работает в связке с другими сервисами Unity:

- Unity Relay — туннелирование сетевого трафика через сервера Unity
- Unity Lobby — поиск и присоединение к играм
- Unity Authentication — авторизация игроков
- Unity Matchmaker (beta) — подбор игроков

Unity Netcode for GameObjects (NGO)

Плюсы

- Официальный и поддерживаемый Unity — обновляется вместе с движком
- Интеграция с Unity Gaming Services
- Простота для начинающих
- Совместим с Addressables, Scenes, Animator и стандартными компонентами
- Открытый исходный код — можно кастомизировать поведение

Минусы

- Не рассчитан на массовые MMO-нагрузки
- Host-based модель может быть нестабильной при плохом соединении
- Нет встроенного dedicated server manager, требуется ручная настройка
- Некоторые возможности (Lobby, Relay) требуют UGS-аккаунта и квот

Unity Netcode for GameObjects (NGO)

Fireworks Mania - An Explosive Simulator		14,99€	93.59%	Dec 2020	12,918	117	1,082
PEAK	-38%	4,64€	92.52%	Jun 2025	135,765	43,433	170,759
Waterpark Simulator		11,99€	91.89%	Aug 2025	17,635	1,539	9,103
Rolling Line		16,79€	91.52%	Apr 2018	6,887	74	215
Sledders		29,99€	91.36%	Mar 2025	9,771	122	1,894
A Gentlemen's Dispute		7,49€	88.72%	Sep 2025	3,434	53	1,015
Haste		18,99€	88.54%	Apr 2025	38,435	170	5,312
Totally Accurate Battlegrounds		Free	87.18%	Jun 2018	86,462	139	29,438
Sex With Friends		4,99€	86.89%	Apr 2025	942	1	92
Virtual Skate		19,50€	86.05%	Aug 2025	408	0	22
Two the Top		7,79€	85.20%	Jun 2025	213	0	10
Gang Beasts		18,99€	83.99%	Dec 2017	249,956	224	3,062
Explosive Odds		4,99€	83.65%	Sep 2025	317	0	29

Более 300 игр с UGS в Stream!

https://steamdb.info/tech/SDK/UnityMultiplayerServices/?sort=rating_desc

FishNet

Open Source. Бесплатный. Альтернатива Mirror, производительнее, лучше поддержка предсказаний.

Особенности:

- Tick-based обновление
- Предсказания и исправления состояния (reconciliation)
- Простая интеграция с dedicated server

https://github.com/FirstGearGames/FishNet

FishNet

FishNet работает по классической схеме Client–Server (Dedicated).

- Один экземпляр игры выступает сервером (может быть запущен локально или удалённо)
- Остальные клиенты подключаются через UDP транспорт
- Поддерживаются Dedicated и Host режимы

FishNet

- NetworkManager — центр управления сетью (запуск, остановка, подключение)
- NetworkObject — отмечает объект, который существует в сетевой среде
- NetworkBehaviour — аналог MonoBehaviour, с сетевыми функциями
- Observers — контроль видимости объекта для разных клиентов
- SyncVars / SyncObjects — синхронизация переменных и коллекций
- RPC (ServerRpc / ObserversRpc / TargetRpc) — вызов удалённых методов
- Predictions / Rollbacks — механизмы предсказаний и откатов
- Transport Layer — поддержка разных транспортов: UDP, KCP, LiteNetLib и др.

FishNet


```
using FishNet.Object;
using UnityEngine;

public class PlayerController : NetworkBehaviour
{
    private void Update()
    {
        if (!IsOwner)
            return;

        float move = Input.GetAxis("Horizontal") * Time.deltaTime * 5f;
        transform.Translate(move, 0, 0);

        if (Input.GetKeyDown(KeyCode.Space))
            JumpServerRpc();
    }

    [ServerRpc]
    private void JumpServerRpc()
    {

        transform.position += Vector3.up * 2f;
    }
}
```

- IsOwner гарантирует, что управление идёт только своим объектом
- ServerRpc — вызывается с клиента, выполняется на сервере
- Все изменения автоматически реплицируются другим клиентам

FishNet

Reliable/Unreliable каналы

- FishNet реализует свою транспортную прослойку, поддерживающую надёжные и ненадёжные передачи данных

- Вы можете выбирать, какие RPC или SyncVar должны быть reliable, а какие — unreliable (например, для позиций)

FishNet

Предсказания и откаты (Prediction / Rollback)

Очень мощная система, аналогичная механике AAA-игр:

- Клиент предсказывает результат своих действий (например, движение персонажа)
- Сервер проверяет и при расхождении откатывает к правильному состоянию

Это ключевая особенность FishNet, недоступная “из коробки” в Mirror или NGO.

FishNet

NetworkObserver

Позволяет показывать объект только определённым клиентам. Например, NPC может быть виден только игрокам в радиусе 50 м.

FishNet

Scene Management

FishNet умеет синхронно загружать и выгружать сцены на сервере и клиентах:

- Поддерживает Additive сцены
- Позволяет держать разных игроков в разных зонах мира

FishNet

Lag Compensation

Имеет встроенную систему компенсации лага — сервер “отматывает” состояния игроков, чтобы корректно обработать попадания по противнику, даже если клиент опоздал с пакетом.

FishNet

Поддерживает различные транспорты:

- LiteNetLib
- KCP
- ENet
- Можно написать собственный транспорт

FishNet

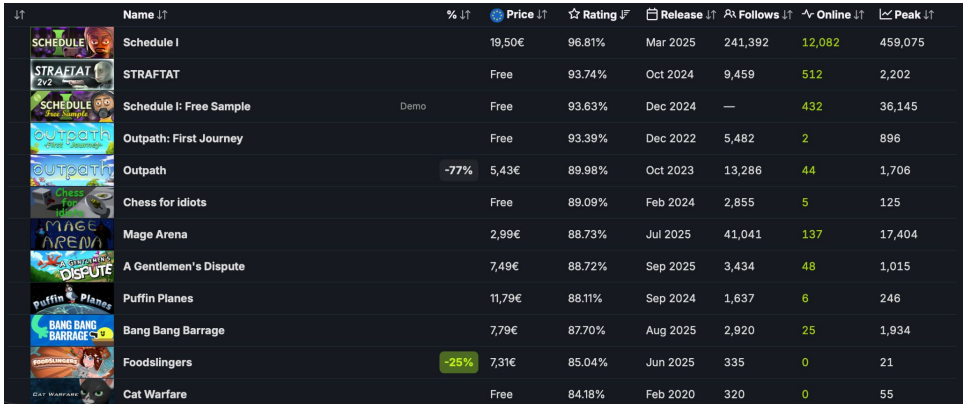
Плюсы

- Поддержка dedicated server и host mode
- Современная архитектура с prediction / rollback
- Встроенные Lag Compensation и Network LOD
- Хорошая производительность и масштабируемость
- Отличная документация и активное сообщество

Минусы

- Более сложный порог входа, чем у Mirror или NGO
- Нет интеграции с Unity Relay/Lobby — нужно решать вручную
- Неофициальный (не от Unity) — могут быть несовместимости с будущими версиями движка
- Меньше туториалов и курсов, чем у Photon/NGO

FishNet



	Name		%	Price	Rating	Release	Follows	Online	Peak
	Schedule I			19,50€	96.81%	Mar 2025	241,392	12,082	459,075
	STRAFTAT			Free	93.74%	Oct 2024	9,459	512	2,202
	Schedule I: Free Sample	Demo		Free	93.63%	Dec 2024	—	432	36,145
	Outpath: First Journey			Free	93.39%	Dec 2022	5,482	2	896
	Outpath		-77%	5,43€	89.98%	Oct 2023	13,286	44	1,706
	Chess for idiots			Free	89.09%	Feb 2024	2,855	5	125
	Mage Arena			2,99€	88.73%	Jul 2025	41,041	137	17,404
	A Gentlemen's Dispute			7,49€	88.72%	Sep 2025	3,434	48	1,015
	Puffin Planes			11,79€	88.11%	Sep 2024	1,637	6	246
	Bang Bang Barrage			7,79€	87.70%	Aug 2025	2,920	25	1,934
	Foodslingers		-25%	7,31€	85.04%	Jun 2025	335	0	21
	Cat Warfare			Free	84.18%	Feb 2020	320	0	55

Около 300 игр на FishNet в Stream!

https://steamdb.info/tech/SDK/FishNet/?sort=rating_desc

LiteNetLib

Низкоуровневая транспортная библиотека. Очень быстрый и лёгкий Reliable UDP. Бесплатная (MIT).

Используется когда нужно:

- Свой собственный неткод
- Минимальные накладные расходы

https://github.com/RevenantX/LiteNetLib

LiteNetLib

LiteNetLib предоставляет минималистичный, но надёжный сетевой слой поверх UDP с:

- поддержкой надежной доставки (Reliable UDP)
- пакетированием и буферизацией
- очередями событий (event-driven)
- и встроенным механизмом пинга, latency и RTT

Это не игровой фреймворк, а “чистый транспорт”. Вы можете строить на нём любой сетевой стек: от чата до MMO.

LiteNetLib

Варианты использования в Unity:

1. Прямое использование — можно импортировать пакет LiteNetLib и написать собственный сетевой слой (для простых игр, чатов, обмена данными)
2. Как транспорт — для Mirror, FishNet, или кастомных решений
3. Для выделенного сервера — сервер можно собрать как headless .NET Core приложение.

LiteNetLib

Плюсы

- Простая API, лёгкая интеграция
- Очень высокая производительность
- Поддержка Reliable/Unreliable, Ping, Disconnect
- Можно встроить собственное шифрование
- MIT License — полностью бесплатна

Минусы

- Нет автоматической сериализации Unity-объектов
- Нет “высокоуровневой логики” (спавн, RPC, SyncVar — нужно писать самому)
- Нет встроенной NAT-проходки или relay-сервисов
- Не предназначена для P2P “из коробки” — только Client–Server

Colyseus

Open-source фреймворк для мультиплеерных игр, написанный на Node.js (TypeScript/JavaScript).

Использует классическую модель “сервер управляет состоянием”:

- Сервер создаёт Room (комнату) для игроков
- В каждой комнате есть объект состояния (state), который описывает игровые данные (позиции, здоровье, очки и т.п.)
- При изменении state сервер автоматически сериализует и синхронизирует эти изменения с подключёнными клиентами (через бинарный протокол на WebSocket)
- Клиенты получают дельты изменений (а не полный state)

https://github.com/colyseus/colyseus

Colyseus


```
import { Room, Client } from "colyseus";

interface Player {
  x: number;
  y: number;
}

export class GameRoom extends Room {
  onCreate(options: any) {
    this.setState({ players: {} });

    this.onMessage("move", (client, data) => {
      const player = this.state.players[client.sessionId];
      player.x += data.x;
      player.y += data.y;
    });
  }

  onJoin(client: Client) {
    this.state.players[client.sessionId] = { x: 0, y: 0 };
    console.log(client.sessionId, "joined");
  }

  onLeave(client: Client) {
    delete this.state.players[client.sessionId];
  }
}
```

Пример сервера

Colyseus


```
using Colyseus;
using UnityEngine;

public class GameClient : MonoBehaviour
{
    public async void Start()
    {
        var client = new ColyseusClient("ws://localhost:2567");
        var room = await client.JoinOrCreate<Room<ExampleState>>("game_room");

        room.OnStateChange += (state, isFirstState) =>
        {
            foreach (var player in state.players)
            {
                Debug.Log($"{player.Key}: {player.Value.x}, {player.Value.y}");
            }
        };


        room.Send("move", new { x = 1, y = 0 });
    }
}
```

Пример клиента на Unity

Colyseus

- State Sync — автоматическая синхронизация состояния между клиентами
- WebSocket транспорт — используется двустороннее соединение на WS
- Serializer Schema — компрессия данных при передаче состояния
- Сообщения — клиент ↔ сервер события (onMessage / send)
- Rooms / Matchmaking — простая логика подбора игроков
- Scalability — поддержка кластеризации через Redis и процесс-менеджеры
- Unity SDK — официальный пакет для клиента на C#
- Web / JS клиенты — можно писать браузерные версии игр

Colyseus

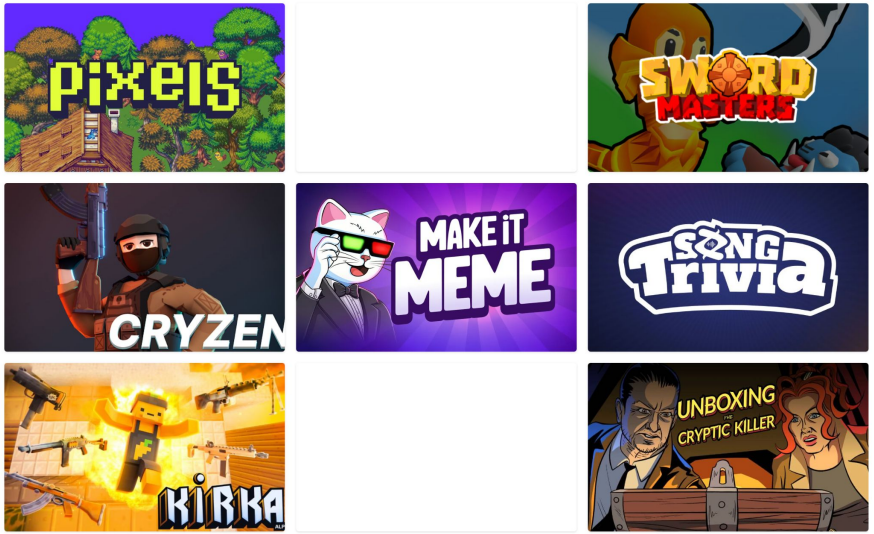
Плюсы

- Простая и чистая архитектура (Room + State)
- Быстрая настройка, особенно если ты знаком с Node.js
- Поддержка Unity, браузера, Godot, Cocos2D и других движков
- Бесплатный и open-source
- Отлично подходит для инди-проектов, прототипов и веб-игр
- Можно разворачивать свой сервер

Минусы

- Нет встроенного UDP — только WebSocket (т.е. TCP)
- Не рассчитан на крупные AAA-игры с большим трафиком
- Нужно писать бэкенд-логику на Node.js
- Нет из коробки готовой системы авторизации, матчмейкинга или античита

Colyseus



https://youtu.be/3EEw5SI93KE

Shardy

Open-source фреймворк для мультиплеерных приложений и игр, написанный на Node.js и TypeScript.

Предоставляет базовую функциональность для построения микросервисных решений:

- мобильных
- социальных
- веб
- многопользовательских игр
- приложений реального времени
- чатов
- middleware сервисов и т.п.

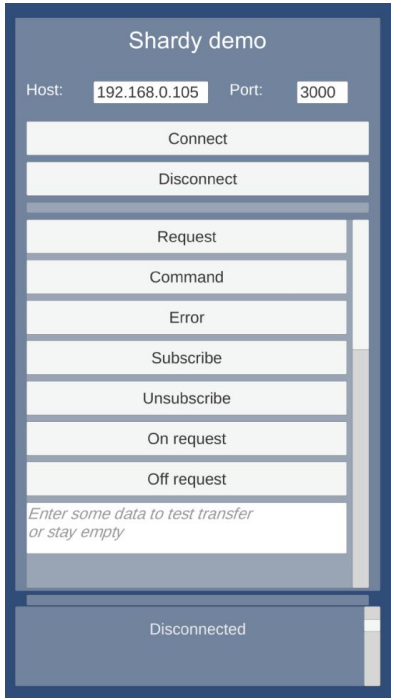
https://github.com/mopsicus/shardy

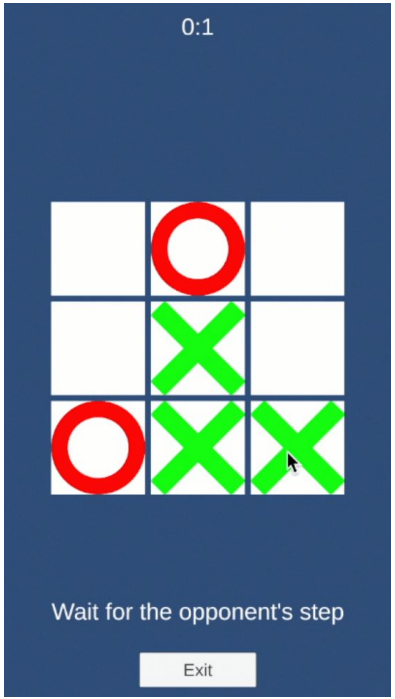
Shardy

Особенности:

- Микросервисная парадигма
- Простой API: запросы, команды, подписки и т.п.
- Транспорт данных через сокеты и вебсокеты
- Легкость и быстрота: Node.js и TypeScript
- Поддержка пользовательской сериализации
- Поддержка пользовательской валидации рукопожатий (handshake)
- Продвинутый логгер: теги, фильтры, области
- Гибкое расширение
- Справочные материалы: документация, сниппеты, примеры
- Почти нулевая конфигурация

Shardy





Shardy

Демо Крестики-Нолики по сети, исходный код сервера и клиента

Shardy

```ts
import { TransportType, Service, Client } from 'shardy';

export class MyService implements Service {

    //
    //  добавляйте свои объекты, базы данных или что-нибудь ещё в этот класс
    //

    // укажите название сервиса
    name = process.env.SERVICE_NAME;

    // укажите тип транспорта
    transport = process.env.SERVICE_TRANSPORT as TransportType;

    async onConnect(client: Client): Promise<void> {
    // новый клиент подключился
    }

    async onDisconnect(client: Client): Promise<void> {
    // клиент отключился
    }

    async onReady(client: Client): Promise<void> {
    // клиент готов к работе
    }

    async onListening(): Promise<void> {
    // сервис запущен
    }

    async onError(error: Error): Promise<void> {
    // произошла какая-то ошибка
    }

    async onClose(): Promise<void> {
    // сервис остановлен
    }

}
```

Пока только голый фреймворк без полезной нагрузки 🥲

Shardy

Плюсы

- Простая и чистая архитектура
- Простой API
- Микросервисная парадигма
- Поддержка Unity, WebGL
- Бесплатный и open-source
- Пользовательская сериализации и handshake
- Можно разворачивать свой сервер

Минусы

- Нет встроенного UDP — только TCP
- Нет из коробки готовой системы авторизации, матчмейкинга или античита
- Нужно писать бэкенд-логику на Node.js и Typescript??
- Пока только транспорт, нет полезной нагрузки в виде модулей

Shardy

Make Shardy great again!

Интеграция SDK

SDK (Software Development Kit) — набор инструментов, библиотек и документации, который позволяет разработчику интегрировать сторонние сервисы или функциональность в своё приложение.

Интеграция SDK

- Монетизация: Unity Ads, AdMob
- Аналитика: Firebase, GameAnalytics
- Авторизация: Google / Facebook
- Внутриигровые покупки: Unity IAP
- Push-уведомления: OneSignal, Firebase

Интеграция SDK

- Библиотеки — .dll (для Unity), .aar (Android), .framework (iOS)
- Скрипты — классы и API для вызова функций из Unity
- Конфигурационные файлы — AndroidManifest.xml, Info.plist, .gradle
- Документация и примеры — как вызывать API, порядок инициализации

Интеграция SDK

- Unity Ads — простая интеграция, встроен в Unity, для начала, если нет медиатора
- Google AdMob — поддержка баннеров, rewarded и interstitial, для Android и iOS, если нужна сеть Google
- AppLovin / IronSource — медиаторы, подключают сразу несколько рекламных сетей, для оптимизации дохода

Интеграция SDK

- Unity Analytics — простая интеграция, статистика по сессиям
- Firebase Analytics — мощный, гибкий, связка с Crashlytics
- GameAnalytics — специализирован под игры, имеет удобный дашборд
- AppMetrica — аналог FB от Яндекс
- DevToDev — специализирован под игры, сложные отчёты

Интеграция SDK

- Facebook SDK — авторизация, шаринг, аналитика
- VK SDK — авторизация, шаринг, сообщения
- PlayFab / Nakama — онлайн-сервисы, лидерборды, бэкенд
- Google/Apple/Huawei/Yandex Sign-In / — авторизация на мобильных платформах

Интеграция SDK

- Unity IAP — Поддержка App Store и Google Play
- RevenueCat SDK — Кроссплатформенное управление подписками
- XSolla — кроссплатформенное решение для покупок, поддержка Web

Интеграция SDK. Проблемы

- Конфликты зависимостей

- Что видим: ошибки сборки Gradle
- Причина: несовместимые библиотеки
- Решение: использовать *External Dependency Manager*

Интеграция SDK. Проблемы

- Дубликаты AndroidManifest.xml

- Что видим: manifest merge failed
- Причина: несколько SDK прописали одинаковые активности
- Решение: использовать tools:replace

Интеграция SDK. Проблемы

- Duplicate class

- Что видим: Duplicate class X found in modules
- Причина: SDK тянут разные версии Google Play Services
- Решение: удалить дубликаты или синхронизировать версии

Интеграция SDK. Проблемы

- SDK не работает на устройстве

- Что видим: нет логов, не выполняется инициализация
- Причина: требуются нативные зависимости
- Решение: проверить, что плагины активны для платформы

Интеграция SDK. UnityAds

```
using UnityEngine;
using UnityEngine.Advertisements;

public class AdsInitializer : MonoBehaviour, IUnityAdsInitializationListener
{
    [SerializeField] string androidGameId;
    [SerializeField] string iosGameId;
    [SerializeField] bool testMode = true;
    string gameId;

    void Awake() => InitializeAds();

    void InitializeAds()
    {
        gameId = (Application.platform == RuntimePlatform.IPhonePlayer)
            ? iosGameId : androidGameId;

        Advertisement.Initialize(gameId, testMode, this);
    }

    public void OnInitializationComplete() => Debug.Log("Ads Initialized");
    public void OnInitializationFailed(UnityAdsInitializationError error, string message)
        => Debug.LogError($"Ads failed: {error} - {message}");
}
```

Интеграция SDK. UnityAds

```
using UnityEngine;
using UnityEngine.Advertisements;

public class RewardedAds : MonoBehaviour, IUnityAdsLoadListener, IUnityAdsShowListener
{
    [SerializeField] string adUnitId = "Rewarded_Android";

    void Start() => LoadAd();

    public void LoadAd() => Advertisement.Load(adUnitId, this);

    public void ShowAd() => Advertisement.Show(adUnitId, this);

    public void OnUnityAdsAdLoaded(string id) => Debug.Log("Ad Loaded");
    public void OnUnityAdsShowComplete(string id, UnityAdsShowCompletionState state)
    {
        if (state == UnityAdsShowCompletionState.COMPLETED)
            Debug.Log("Reward player!");
    }

    public void OnUnityAdsFailedToLoad(string id, UnityAdsLoadError error, string message)
        => Debug.LogError(message);
    public void OnUnityAdsShowFailure(string id, UnityAdsShowError error, string message)
        => Debug.LogError(message);
    public void OnUnityAdsShowStart(string id) { }
    public void OnUnityAdsShowClick(string id) { }
}
```

Интеграция SDK. EDM

https://github.com/googlesamples/unity-jar-resolver

```
{
  "dependencies": {
    "com.google.external-dependency-manager": "1.2.178"
  }
}
```

Что дальше

- ИИ для игр
- Сборка под разные платформы