オフラインのUnityゲームをFusionでオンライン化してみた

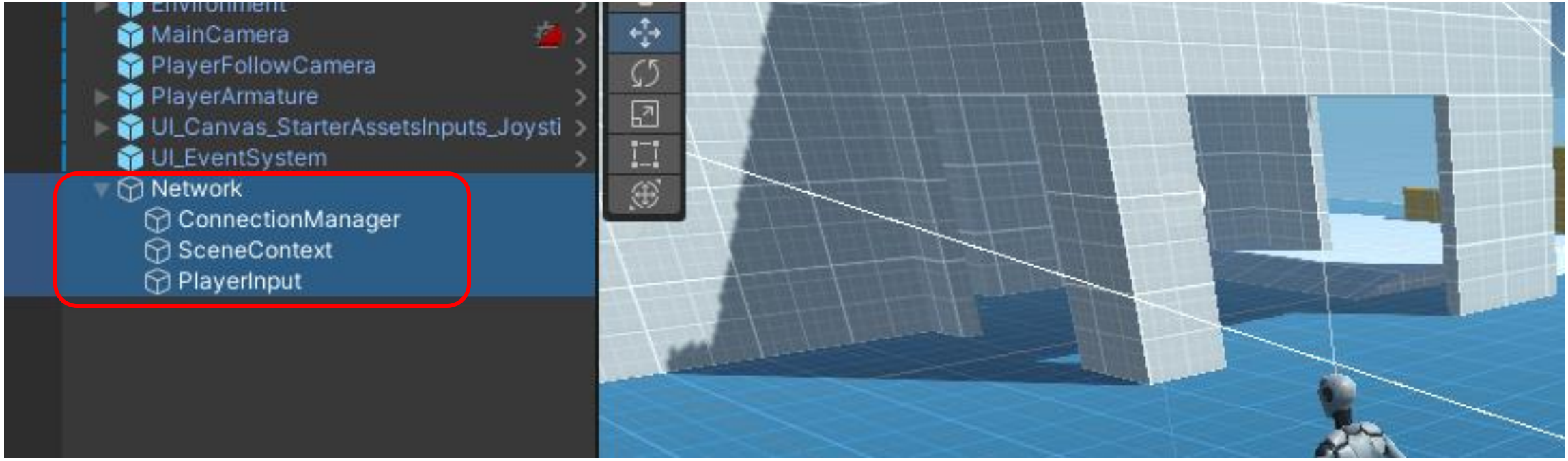
手軽にみなさんのゲームのオンライン化ができるよう Photon Fusion の導入、使い方、仕組みを解説します

手軽にオフラインのゲームをオンライン化の概要

シーンに追加したのは赤枠で示している3つのスクリプトコンポーネントだけ

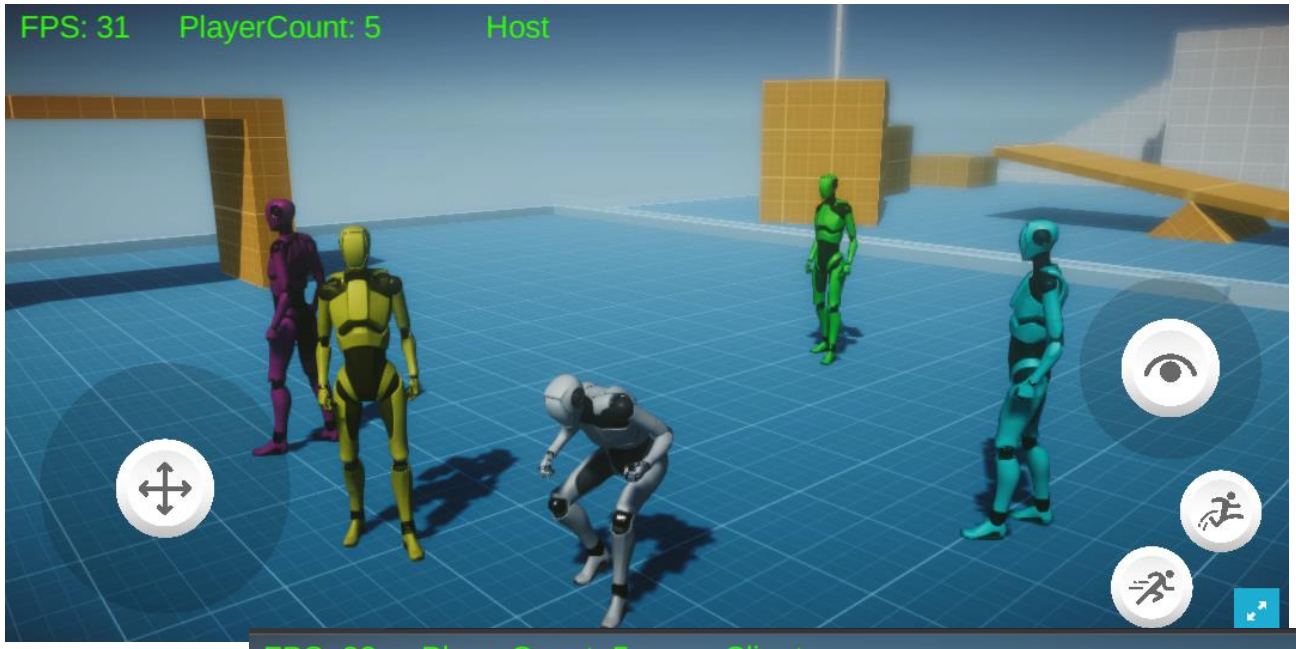
一切オフラインのゲームに変更を加えずにオンライン化

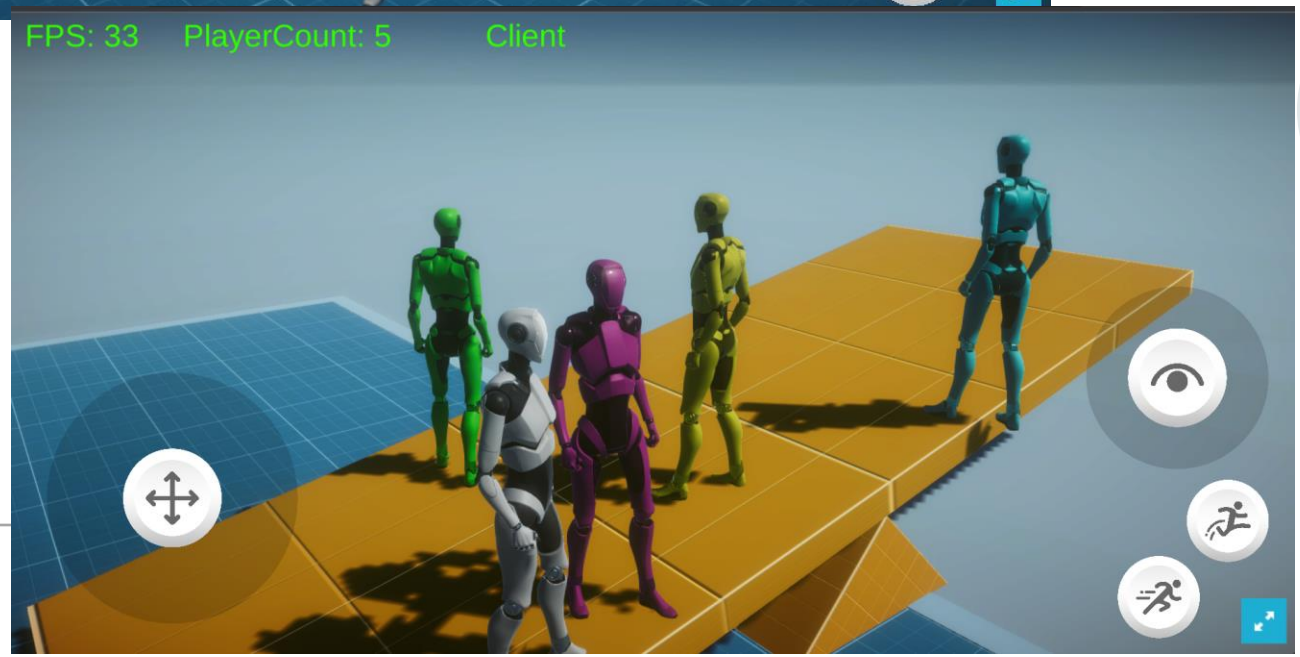
（本発表のここがすごい！）



発表内容

1. Fusion の概要に触れながら、**オフライン部分を一切変更せず、オンライン化する手法を紹介**します（**サンプルコードは公開**しています: URL、実装詳細のメモを取る必要ありません）
2. 休憩：Web公開しているオンラインゲームを皆さんと遊びます（同時 100人まで参加可能）
3. **WebGLビルドを簡単にWebで公開する手順を紹介**します





自己紹介

シンプルスター@Unityゲーム制作
@lpcwstr

趣味でUnityゲーム開発進捗やTipsを発信してます。たまに興味あるニュースと日々思ったことなど
全プレイヤーでたった一つの世界を共有する3Dサンドボックスゲーム
#CubeArtWorld をリリースして運営中！
無料配信▶ simplestar-game.itch.io/cubeartworld

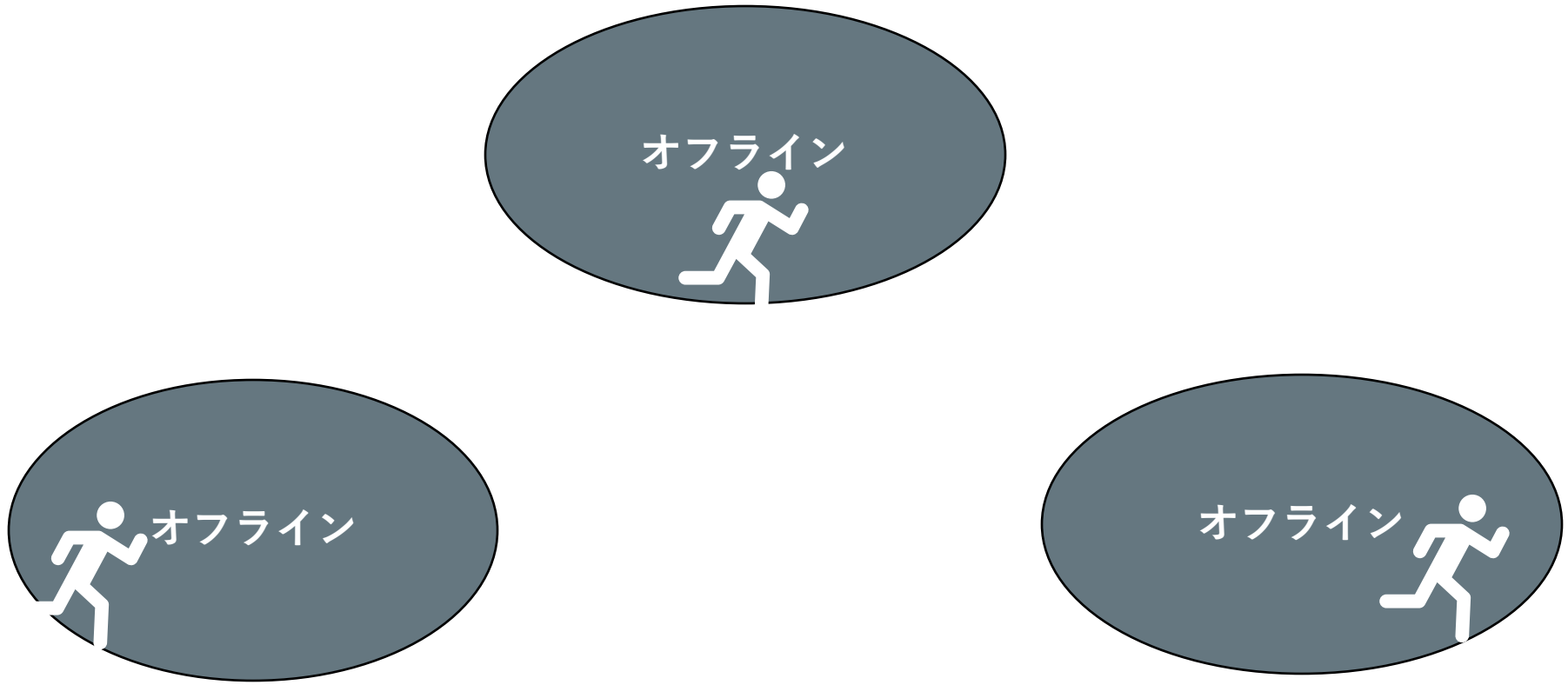
ゲーム開発者 Tokyo youtube.com/channel/UCB8u7...
2009年11月からTwitterを利用しています

421 フォロー中 **3,159** フォロワー

趣味で Unity ゲーム開発しています
普段はこの Twitter アカウントでシンプルスターとしてゲーム進捗を中心につぶやいてます（たまに思ったことなど）
CubeArtWorld というオンライン3Dサンドボックスゲーム: URLを2年運営してます（今はFusion検討中）

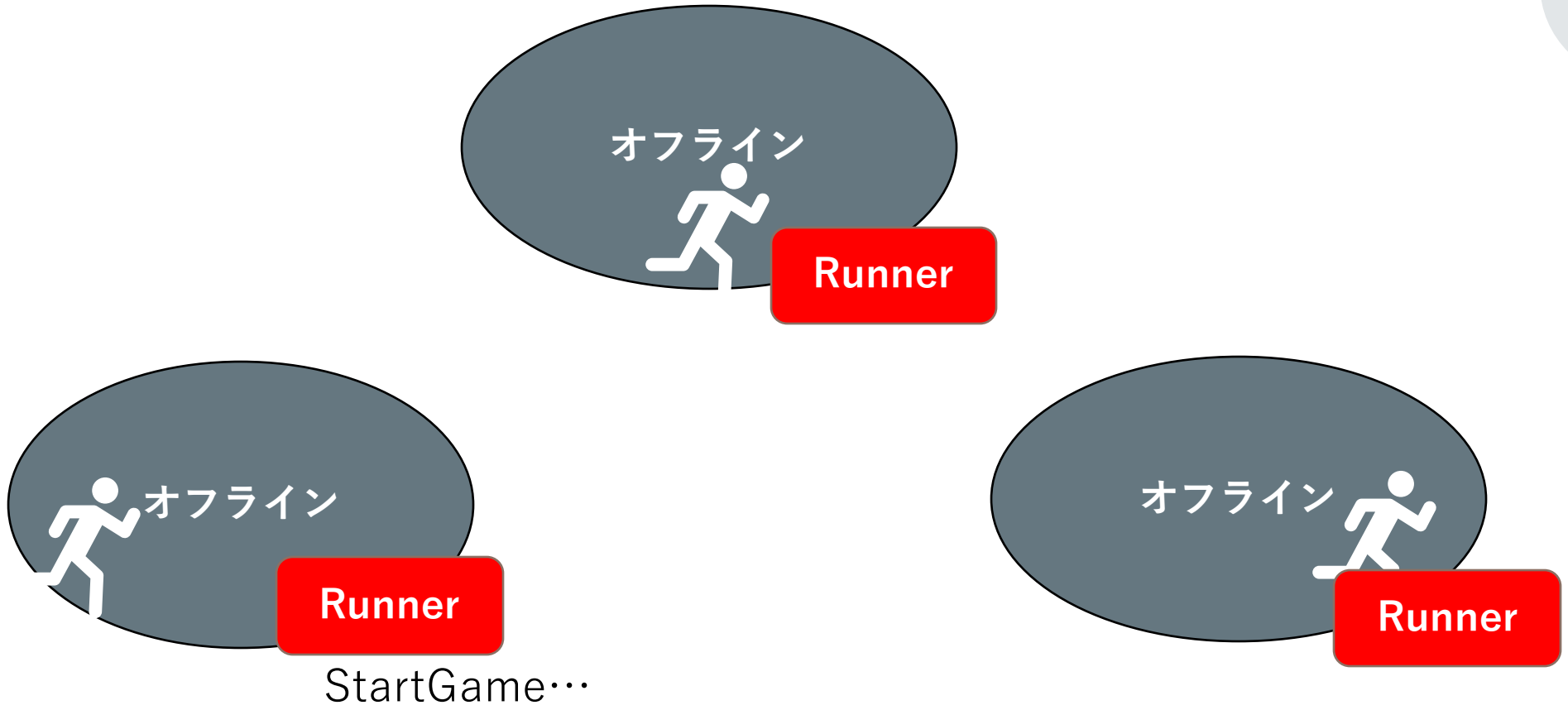
オンライン化のイメージ 1/4

ここにオフラインで遊べる
3つのクライアントゲームがあります



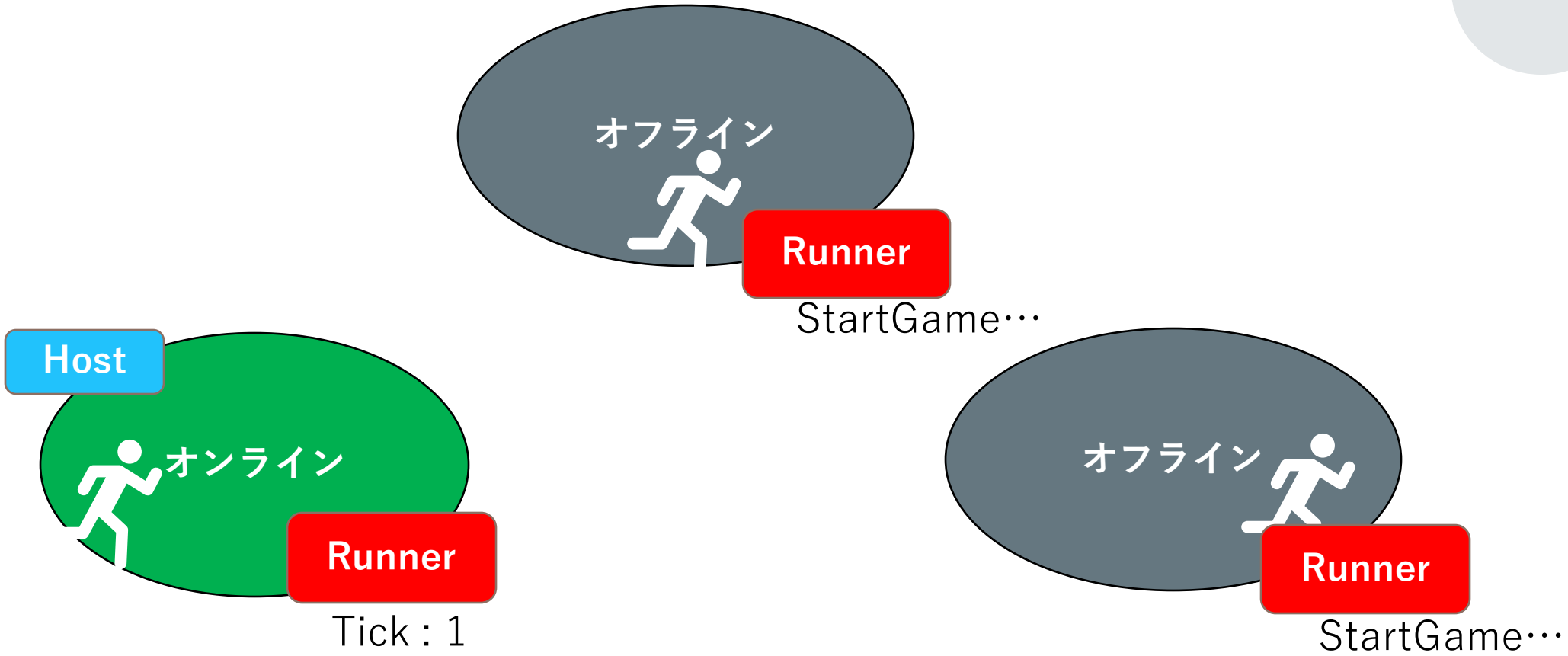
オンライン化のイメージ 2/4

Fusion では NetworkRunner をシーンに配置し、StartGame 関数を実行してセッションへの参加を試みます



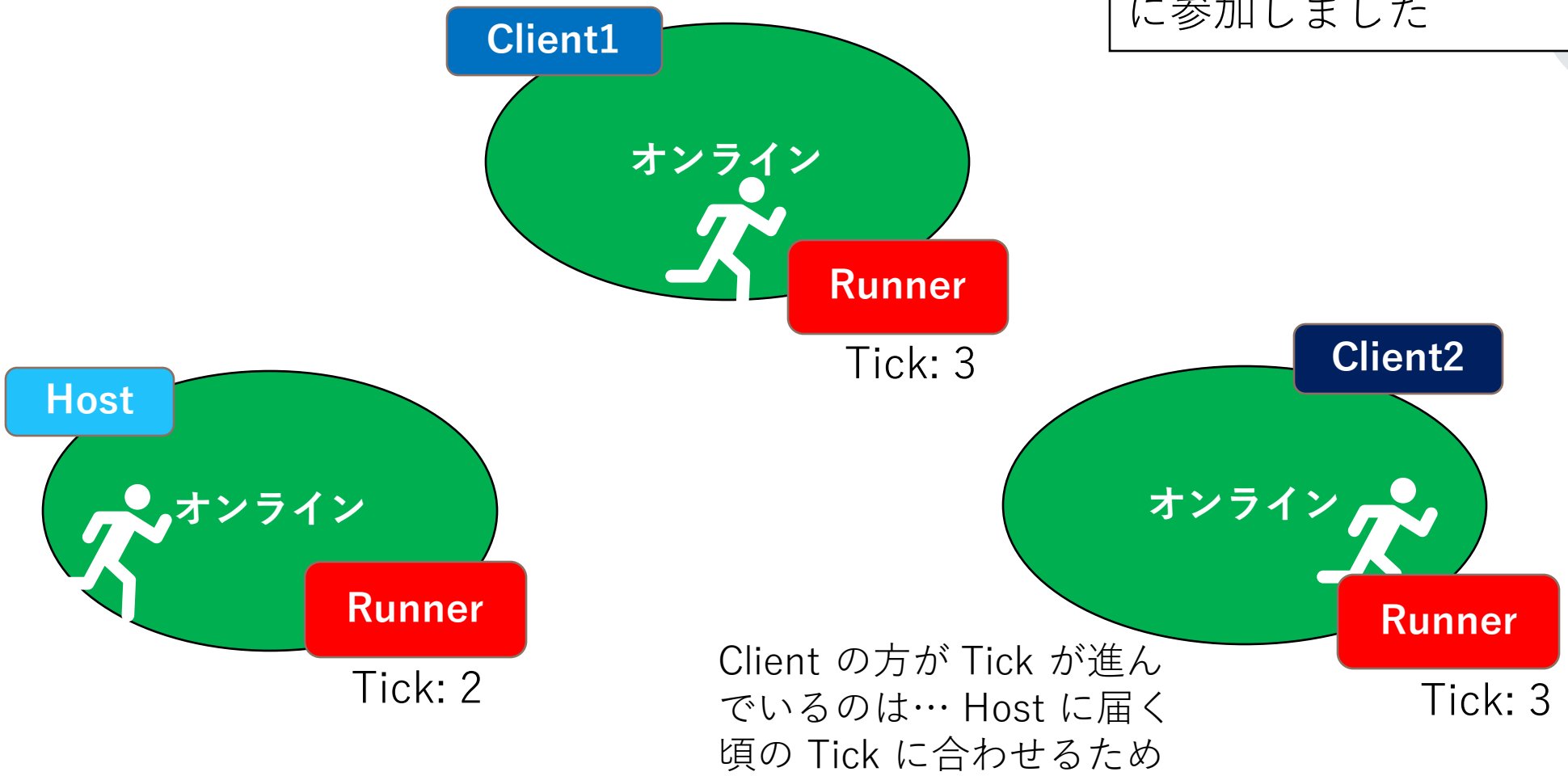
オンライン化のイメージ 3/4

最初にセッションに参加したシーンが Host となりました



オンライン化のイメージ 4/4

後から参加したシーンが Client となりました
すべてのシーンが同じセッションに参加しました



Client の方が Tick が進んでいるのは… Host に届く頃の Tick に合わせるため

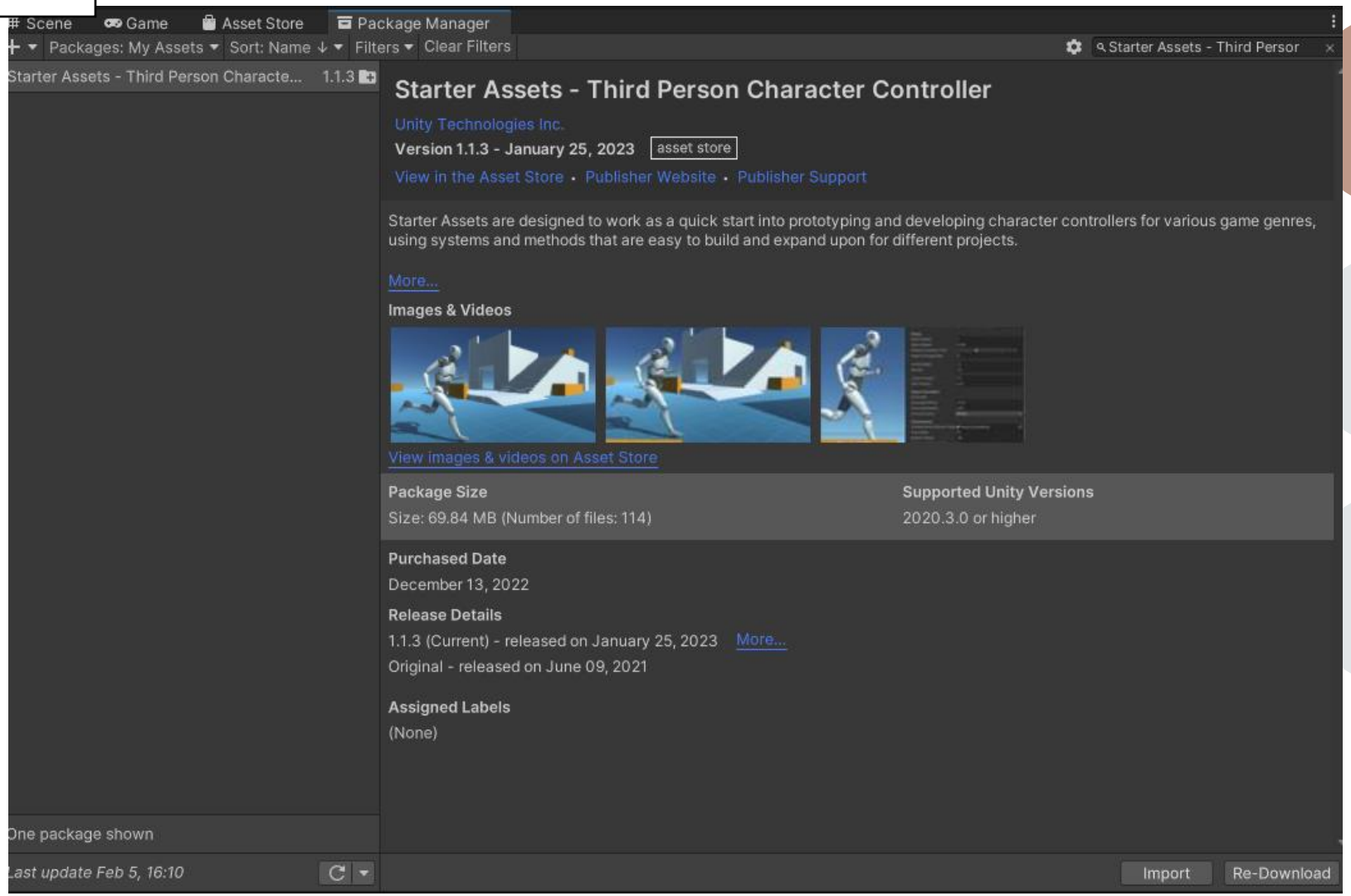
先ほどのイメージを具体的な手順で見ていきましょう

Unity StarterAssetsでオフラインのサンプルシーンを作成する

Unity は 2021.3.19f1 を使用し新規プロジェクトは **3Dコア**をテンプレートに作成し、ビルドターゲットはこの時点で **WebGL** にしておきます

まずは次の [Starter Assets - Third Person Character Controller] アセットを Package Manager からインポートします（無料です）

URP 専用なので Project Settgins の設定更新を忘れずに行います



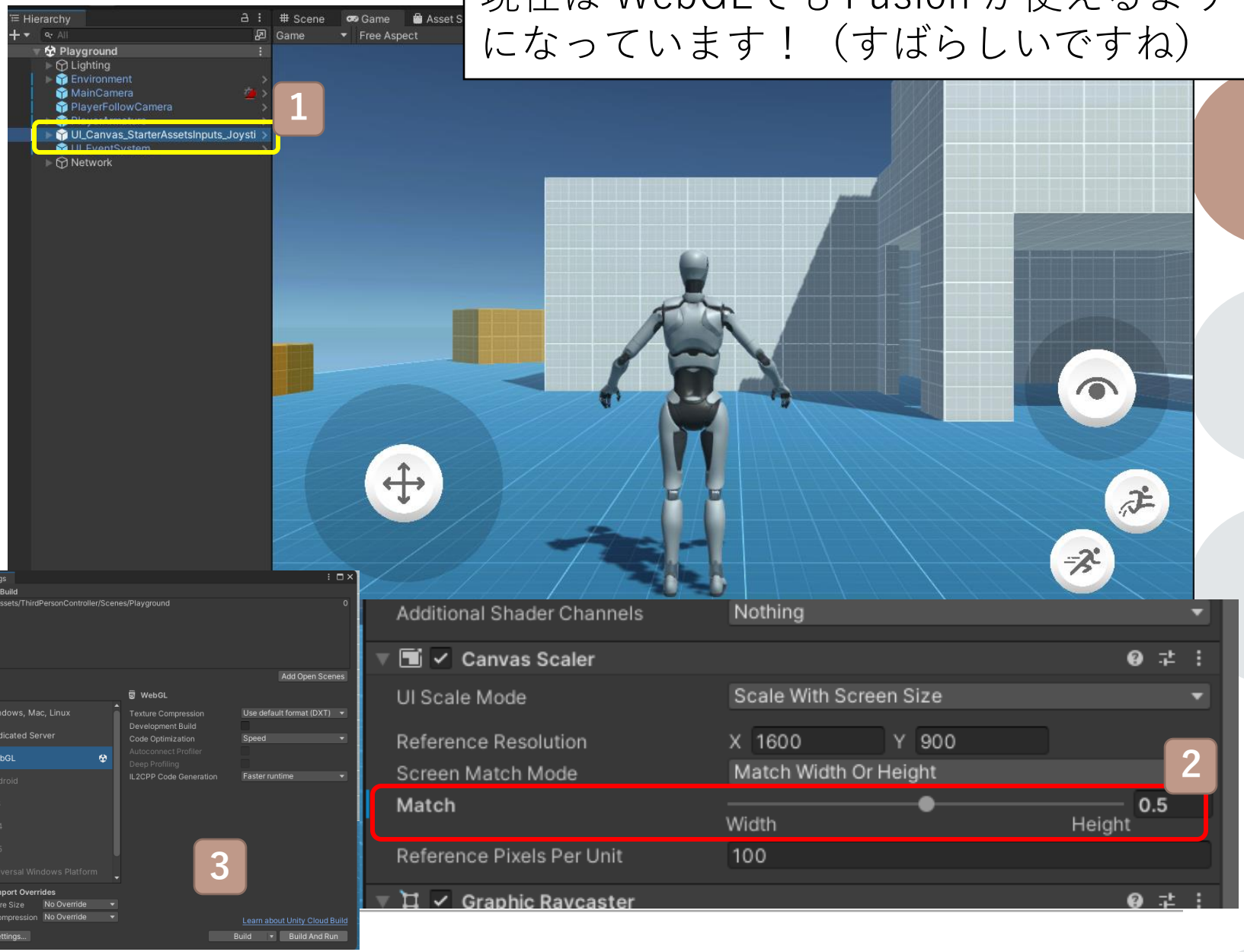
WebGL ビルド確認

1．スマホからも動かせるよう UI ボタンをアクティブにしたいので **黄色枠**のUI用 Game Object をアクティブに切り替えます

2．Canvas Scaler で縦画面や横画面に切り替わっても、いい感じのスケールで UI が表示されるように赤枠の Match を 0.5 に設定します

3．Build Settings で Playground シーンをビルド対象に追加して Build & Run すれば localhost で動作確認できます

現在は WebGLでも Fusion が使えるようになっています！（すばらしいですね）



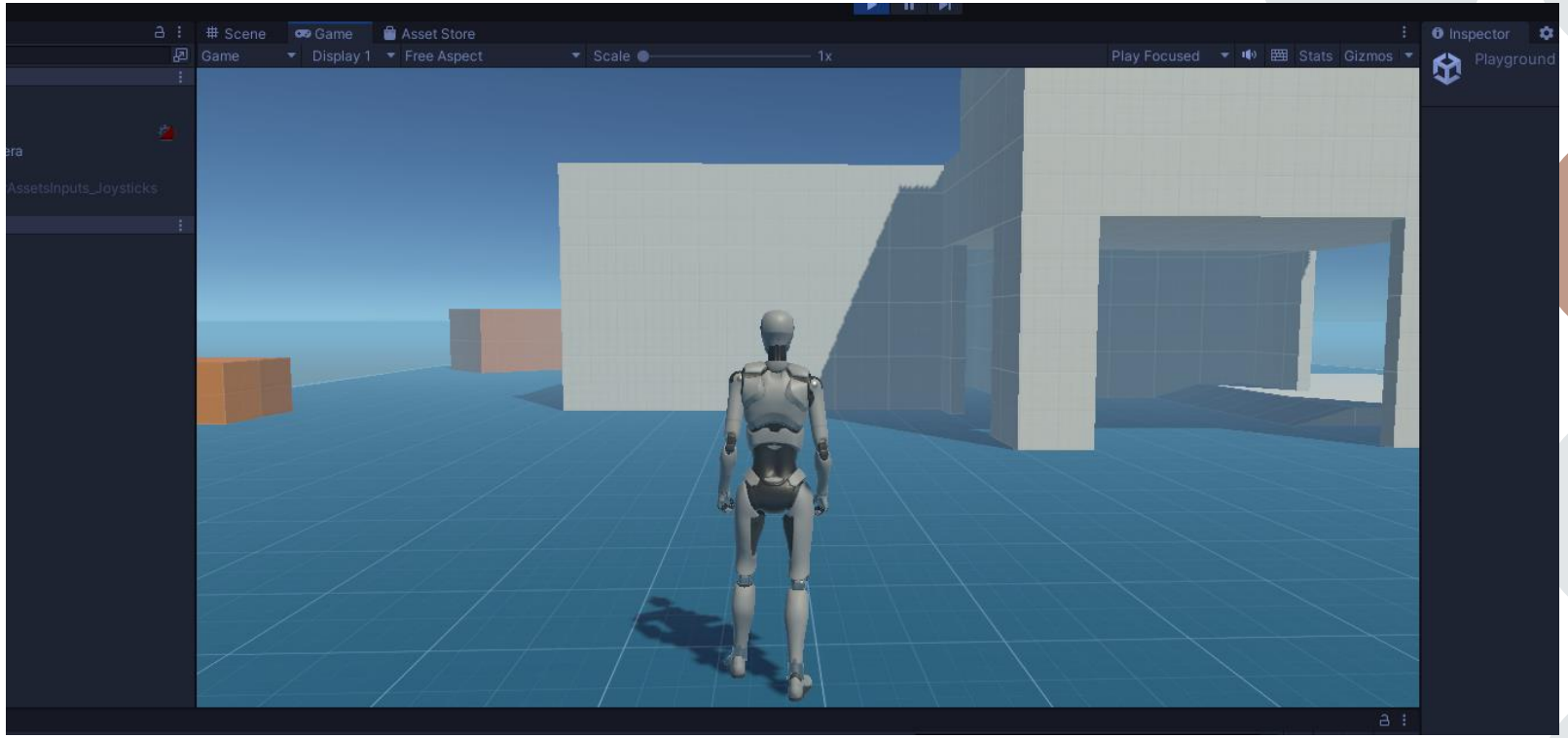
オフラインのゲームが完成！

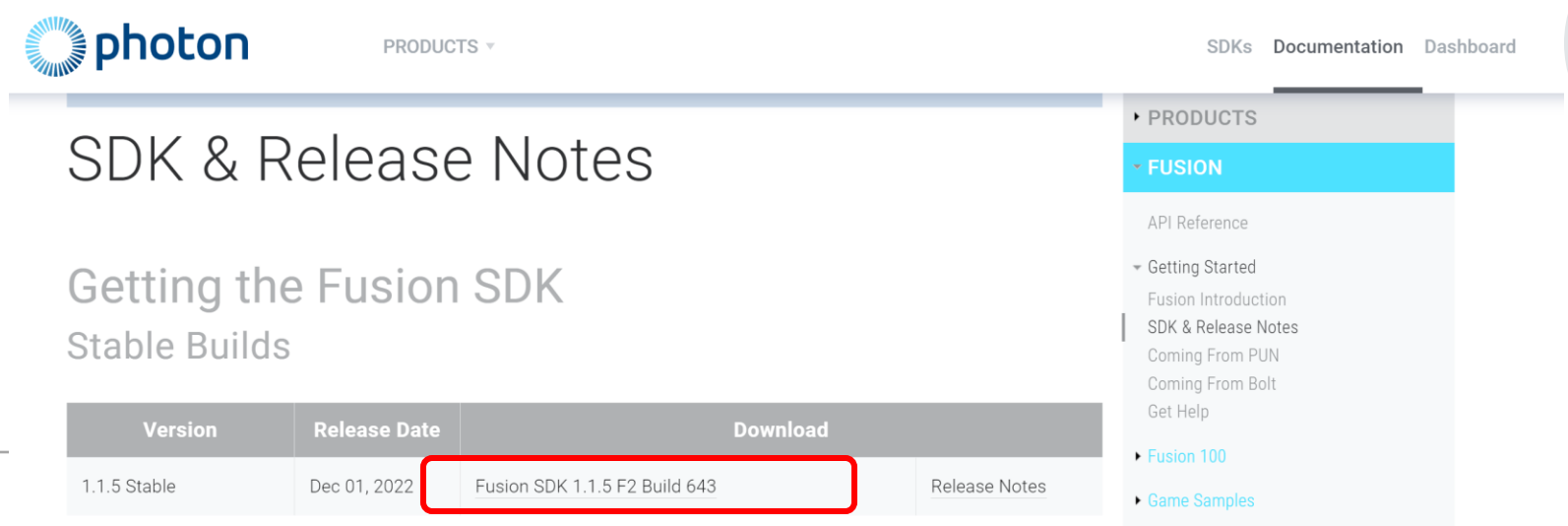
ではゲームをオンライン化していきましょう！

まずは **Photon Fusion SDK 最新版を**[ダウンロード](#)**ページの赤枠をクリックして入手**します

Fusion SDK 1.1.5 F2 Build 643 を今回は使いました

.unitypackage ファイルが得られるので、これをプロジェクトにインポートします



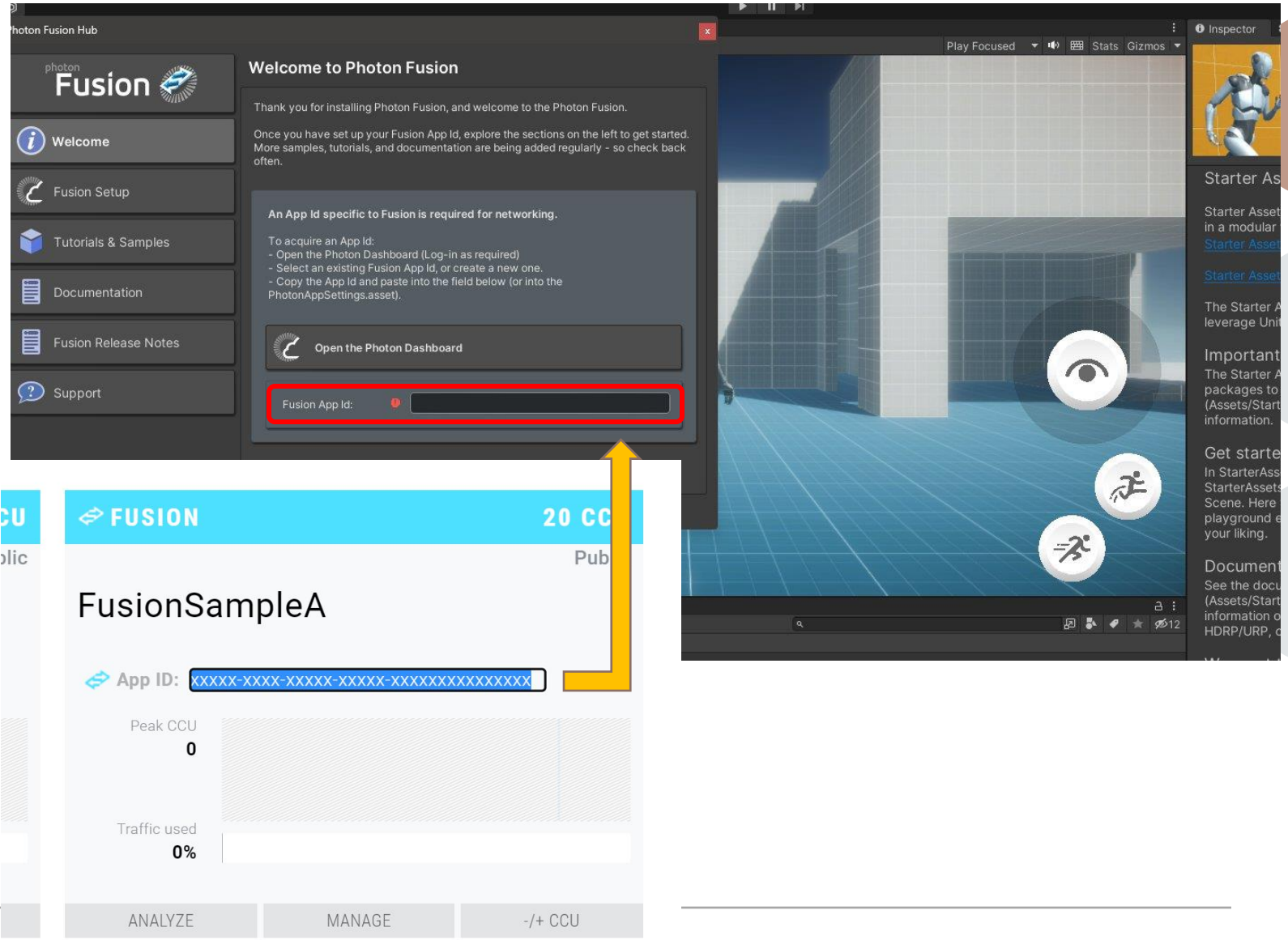


FUSION App ID を Unityプロジェクトに登録

Fusion SDK パッケージをインポート直後、App ID の入力を求められます

SDK のダウンロードページから Dashboard を開けます。そこから自身の FUSION アプリを作成（無料でいくつもアプリが作れます！）アプリに記載されている App ID を**赤枠の欄**に貼り付けます

これでプロジェクトは無料枠の 20 人同時接続まで可能な設定になりました



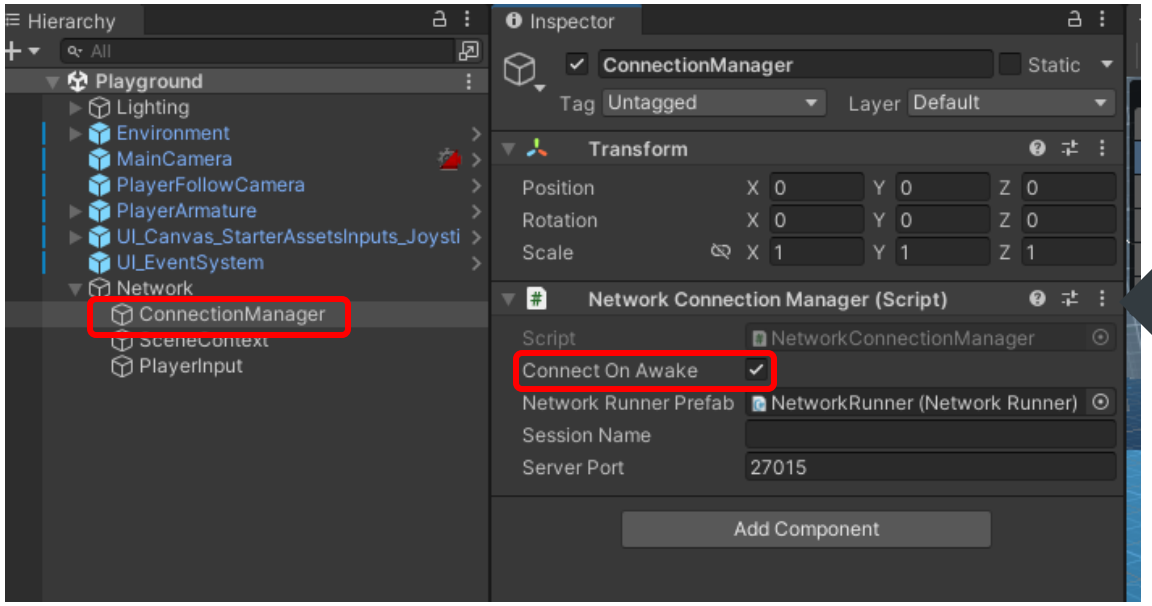
AutoHostOrClient Modeで接続する

(ConnectionManager のお仕事)

起動後すぐにオンライン接続するゲームを作ります

赤い矢印 で示している AutoHostOrClient は、最初に Session に参加したシーンが Host になり、後続で参加したシーンが Client になります

ユーザーは Host や Client を選択するフローをスキップしてオンラインゲームに参加できます

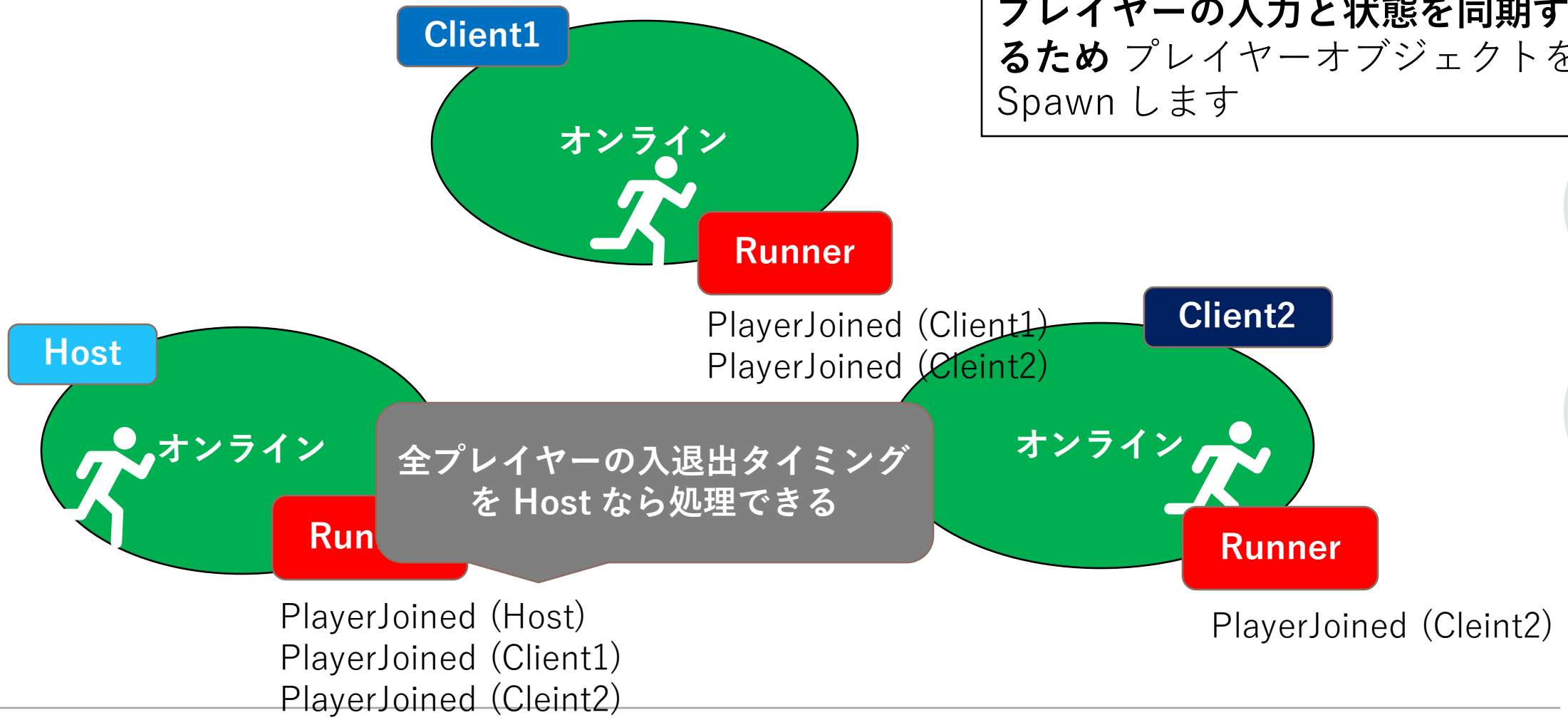


ユーザースクリプト

```
Task InitializeNetworkRunner(NetworkRunner runner)
{
    var sceneManager = runner.GetComponents(typeof(MonoBehaviour)).OfType<INetworkSceneManager>().FirstOrDefault();
    if (null == sceneManager)
    {
        sceneManager = runner.gameObject.AddComponent<NetworkSceneManagerDefault>();
    }
    return runner.StartGame(new StartGameArgs
    {
        GameMode = GameMode.AutoHostOrClient,
        Address = NetAddress.Any(this.serverPort),
        SessionName = this.sessionName,
        SceneManager = sceneManager,
    });
}
```

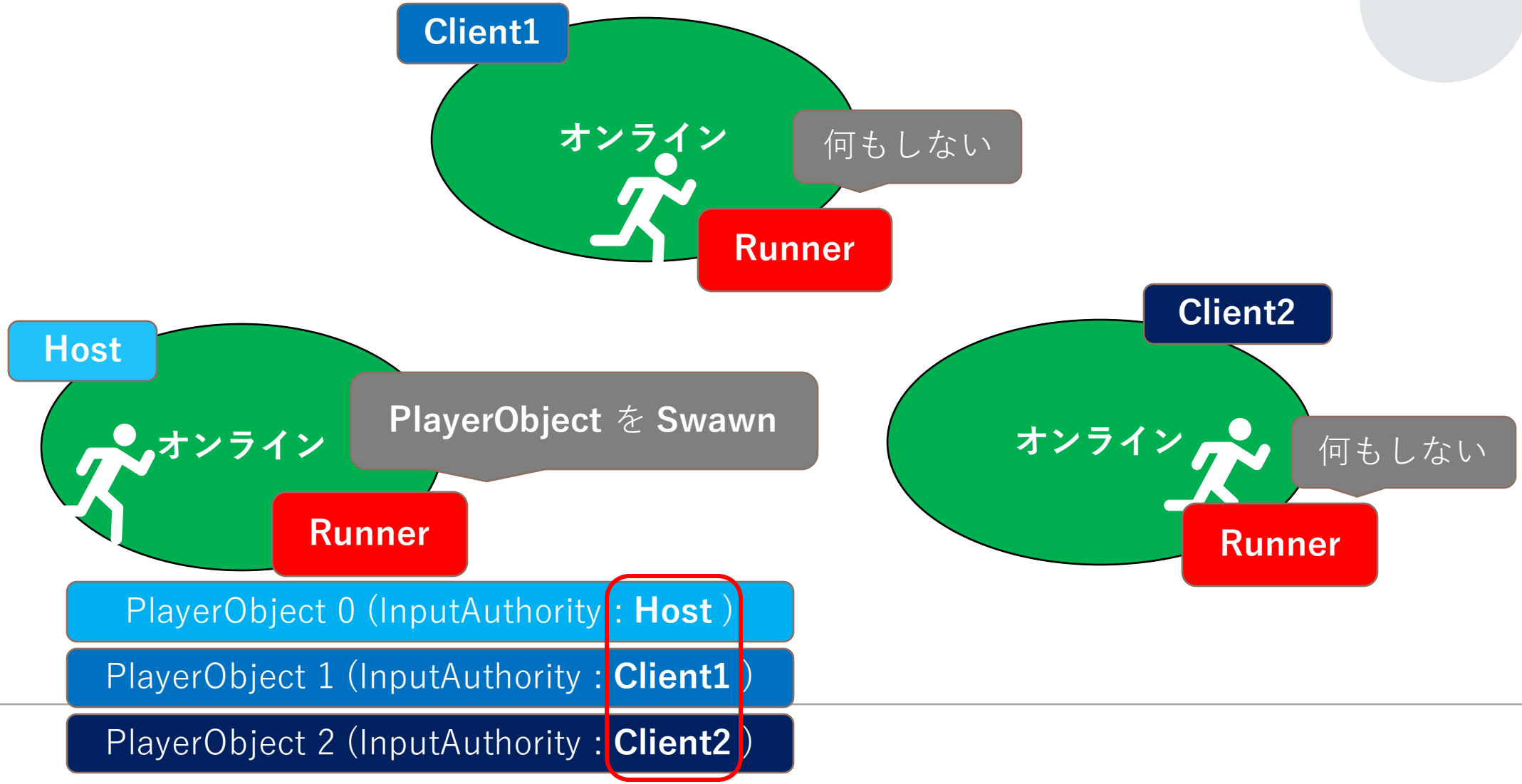
入力と状態の同期 1/5

PlayerJoined 関数がプレイヤーがセッションに参加するたびに呼ばれます
プレイヤーの入力と状態を同期するため プレイヤーオブジェクトを Spawn します



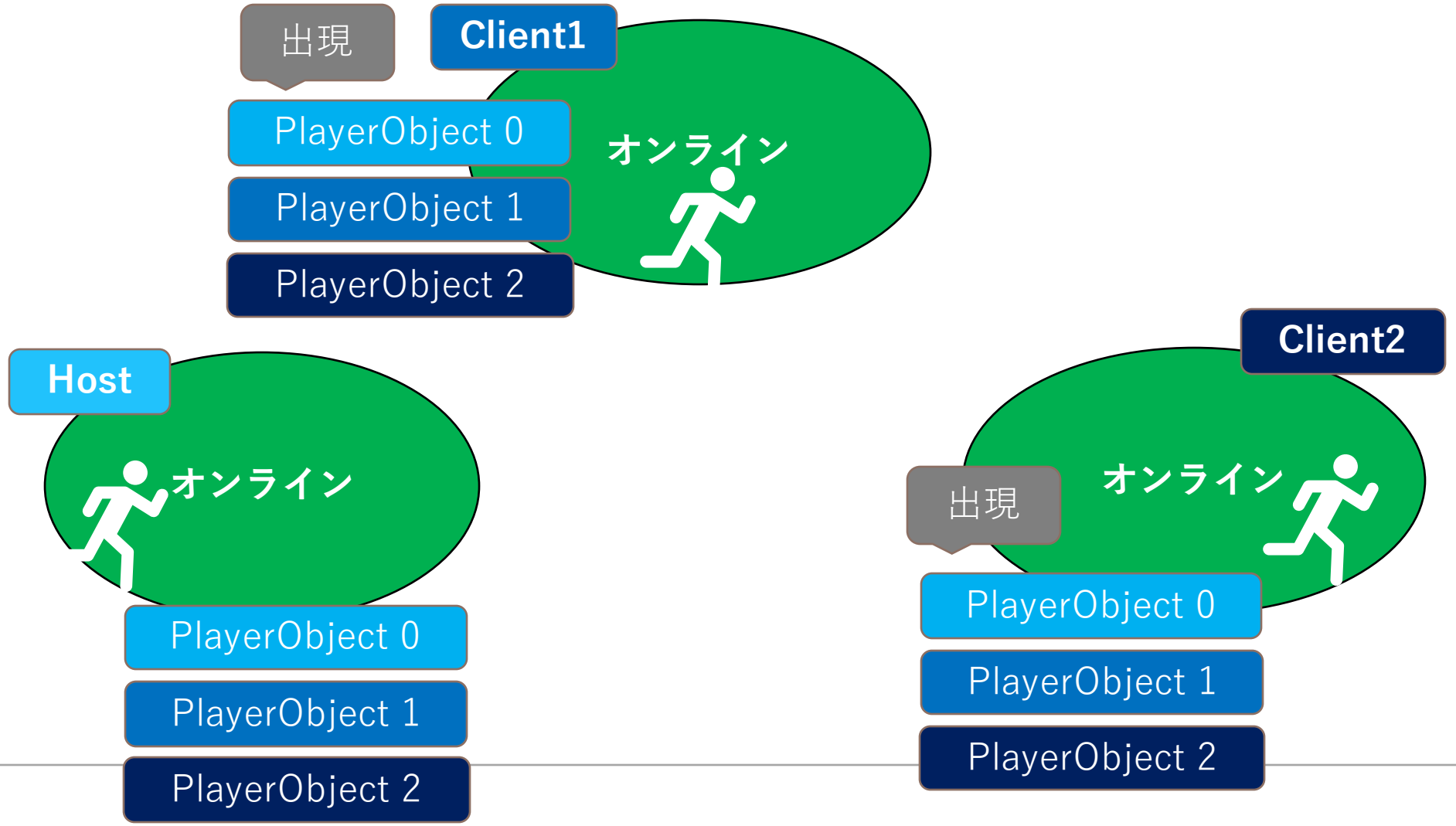
入力と状態の同期 2/5

Spawn する際、**InputAuthority** に参加してきたプレイヤーを設定します



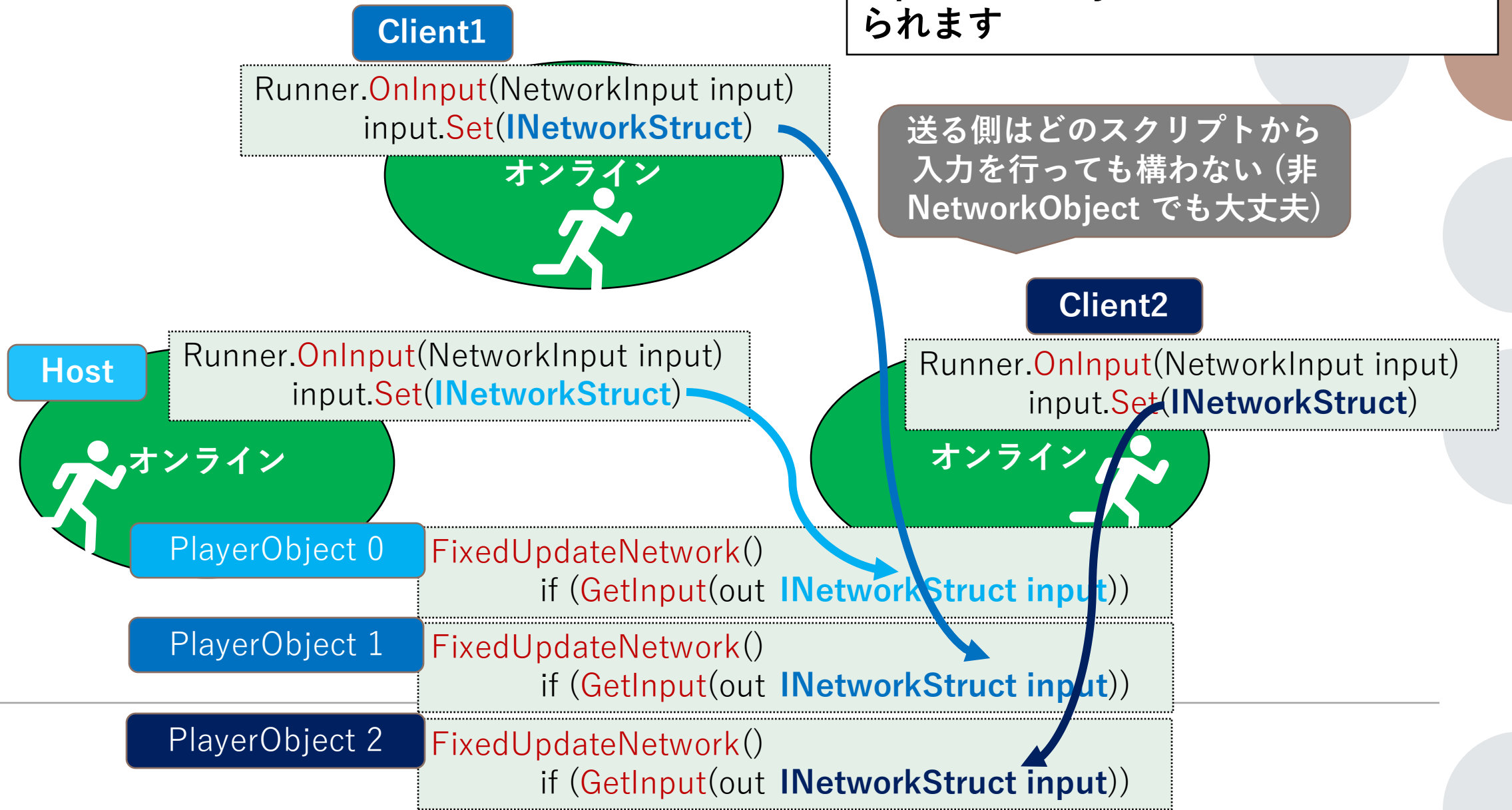
入力と状態の同期 3/5

PlayerObject(Spawn 可能な NetworkObject) は Host で Spawn すると Client シーンにも出現します



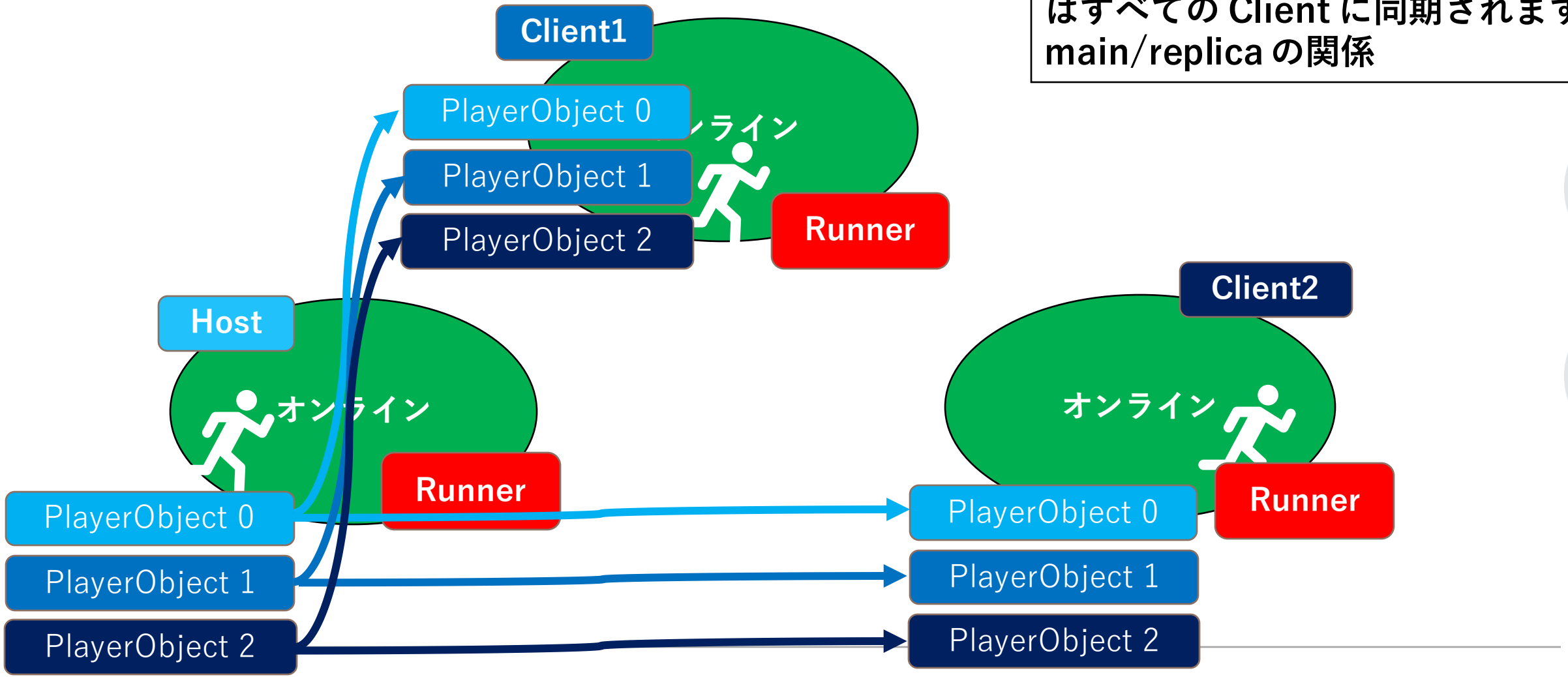
入力と状態の同期 4/5

InputAuthority の理解として、
NetworkObject の **GetInput では、InputAuthority の Client の入力が得られます**



入力と状態の同期 5/5

StateAuthority の理解としては
Host で変更した State
([Networked] 属性の プロパティ)
はすべての Client に同期されます
main/replica の関係

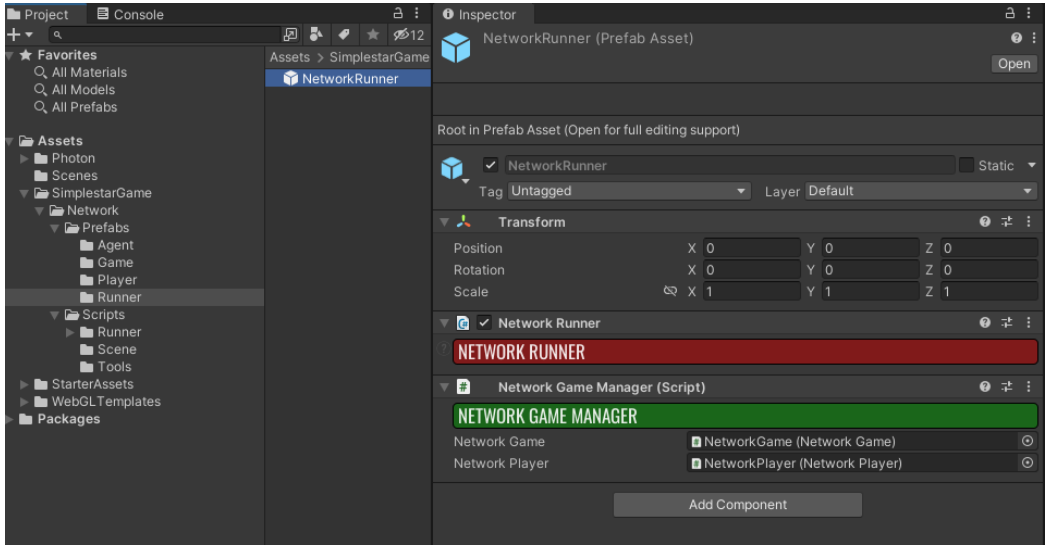


プレイヤーオブジェクトの作成

(NetworkRunner Prefab の詳細)

NetworkRunner の Prefab にアタッチした SimulationBehaviour 継承クラスにて **IPlayerJoined を実装すると、プレイヤーが Session に参加したタイミングで PlayerJoined が呼ばれます**

ここで inputAutority を Join したプレイヤーに設定してプレイヤーオブジェクトを Spawn しています



ユーザースクリプト

```
public class NetworkGameManager : SimulationBehaviour, IPlayerJoined, IPlayerLeft
{
    [SerializeField] NetworkGame networkGame;
    [SerializeField] NetworkPlayer networkPlayer;

    void IPlayerJoined.PlayerJoined(PlayerRef playerRef)
    {
        if (!this.Runner.IsServer)
        {
            return;
        }
        if (!this.networkGameSpawned)
        {
            this.Runner.Spawn(this.networkGame);
            this.networkGameSpawned = true;
        }
        var networkPlayer = this.Runner.Spawn(this.networkPlayer, inputAuthority: playerRef);
        this.networkPlayers.Add(playerRef, networkPlayer);
        this.Runner.SetPlayerObject(playerRef, networkPlayer.Object);
    }
```

Host 意外は何もしない早期リターン
(Host は IsServer フラグが True)

参加してきたプレイヤーを指定

InputAuthority

INetworkRunnerCallbacks を実装したクラスを Runner.AddCallbacks に渡すと **Tick ごとに OnInput が呼ばれるようになります**

NetworkInput に INetworkStruct を継承した構造体を Set します

Host で NetworkBehaviour を継承したクラスにて FixedUpdateNetwork などで Tick ごとに GetInput すると、**InputAuthority に設定された Client の入力を受け取ります**

Cleint シーンの INetworkRunnerCallbacks 実装クラス

```
void INetworkRunnerCallbacks.OnInput(NetworkRunner runner, NetworkInput input)
{
    input.Set(new PlayerInput
    {
        move = this.starterAssetsInputs.move,
        look = this.starterAssetsInputs.look,
        jump = this.starterAssetsInputs.jump,
        sprint = this.starterAssetsInputs.sprint,
        cameraEulerY = this.mainCamera.eulerAngles.y,
        position = this.agentTransform.position,
        rotation = this.agentTransform.rotation,
    });
}
INetworkRunnerCallbacks
```

通信

Host シーンの NetworkBehaviour 継承クラス

```
void IBeforeTick.BeforeTick()
{
    if (GetInput(out PlayerInput input))
    {
        this.Statistics.Position = input.position;
        this.Statistics.Rotation = input.rotation;
        this.StarterAssetsInputs.move = input.move;
        this.StarterAssetsInputs.look = input.look;
        this.StarterAssetsInputs.jump = input.jump;
        this.StarterAssetsInputs.sprint = input.sprint;
        this.StarterAssetsInputs.cameraEulerY = input.cameraEulerY;
        bool agentValid = this.ActiveAgent != null && this.ActiveAgent.Object != null;
        if (agentValid)
        {
            this.ActiveAgent.ApplyInput(input);
        }
    }
}
```

StateAuthority

赤枠で示している
[Networded] 属性を与えたプロパティが**状態(State)**です

Host モードではすべての NetworkObject の **StateAuthority に Host が設定されています**

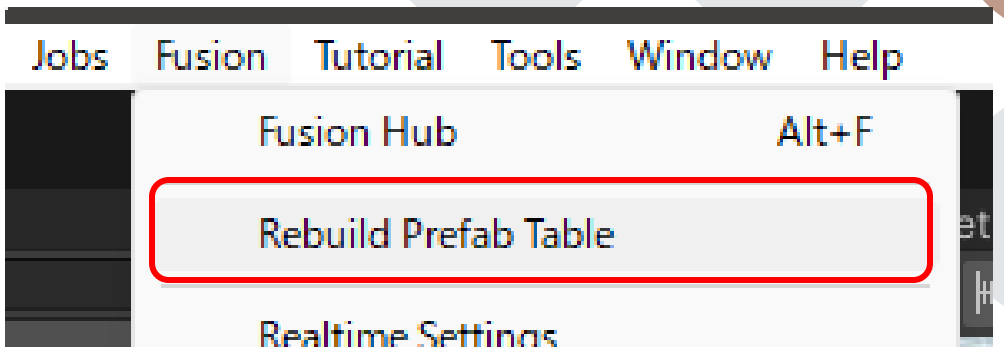
Host にて状態を更新すると、Client にも変更した状態が伝わります（Client にて状態の更新を実装しても処理は無視されます）

OnChanged に関数を指定できるので、状態が変化したタイミングに処理を記述できます（便利）

```
namespace StarterGame
{
    2 個の参照
    public struct PlayerStatistics : INetworkStruct
    {
        public PlayerRef PlayerRef;
        public Vector3 Position;
        public Quaternion Rotation;
    }

    2 個の参照
    public struct StarterAssetInputs : INetworkStruct
    {
        public Vector2 move;
        public Vector2 look;
        public NetworkBool jump;
        public NetworkBool sprint;
        public float cameraEulerY;
    }

    Unity スクリプト (1 件のアセット参照) | 1 個の参照
    public class NetworkPlayer : NetworkBehaviour, IBeforeTick
    {
        [SerializeField] NetworkPlayerAgent agentPrefab;
        1 個の参照
        internal NetworkPlayerAgent AgentPrefab => this.agentPrefab;

        [Networked(OnChanged = nameof(OnActiveAgentChanged), OnChangedTargets = OnChangedTargets.All), HideInInspector]
        16 個の参照
        internal NetworkPlayerAgent ActiveAgent { get; private set; }
        [Networked]
        6 個の参照
        internal ref PlayerStatistics Statistics => ref MakeRef<PlayerStatistics>();
        [Networked]
        11 個の参照
        internal ref StarterAssetInputs StarterAssetsInputs => ref MakeRef<StarterAssetInputs>();

        1 個の参照
        public static void OnActiveAgentChanged(Changed<NetworkPlayer> changed)
        {
```

落とし穴

Prefab Table をリビルドすることを忘れずに！



NetworkObject の Spawn は Prefab を利用しています

Spawn 時にテーブルに Prefab が無いというエラーが出る場合があります

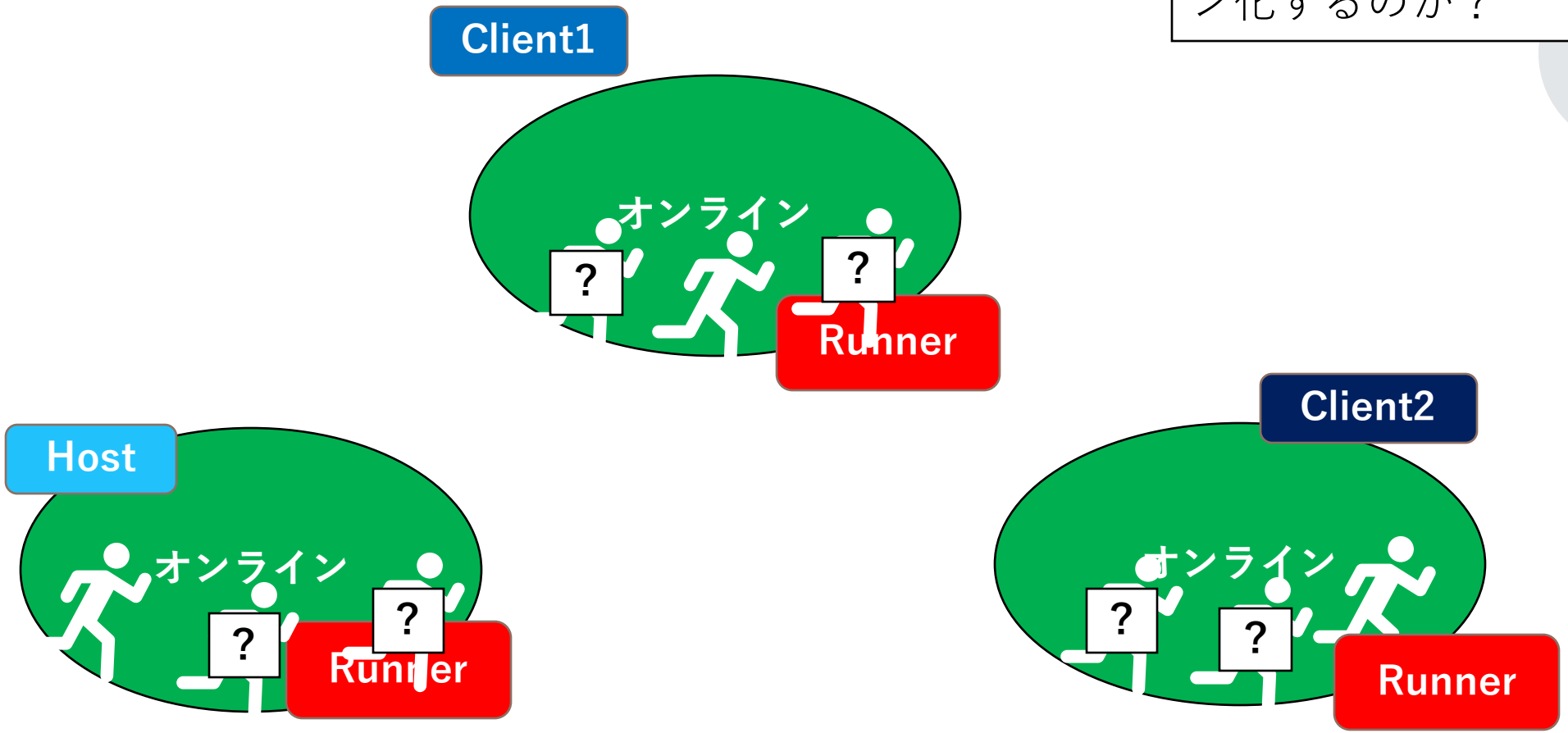
Prefab Table が古い場合はリビルドする必要があるので、忘れずに行っておきましょう

メニューの Fusion > Rebuild Prefab Table

基本のおさらい
Host で Spawn した NetworkObject (Prefab) は Client シーンでも Spawn されます

オフライン実装を変更しない仕組み 1/2

オフラインのゲームを変更することなく、どうやってオンライン化するのか？



オフライン実装を変更しない仕組み 2/2

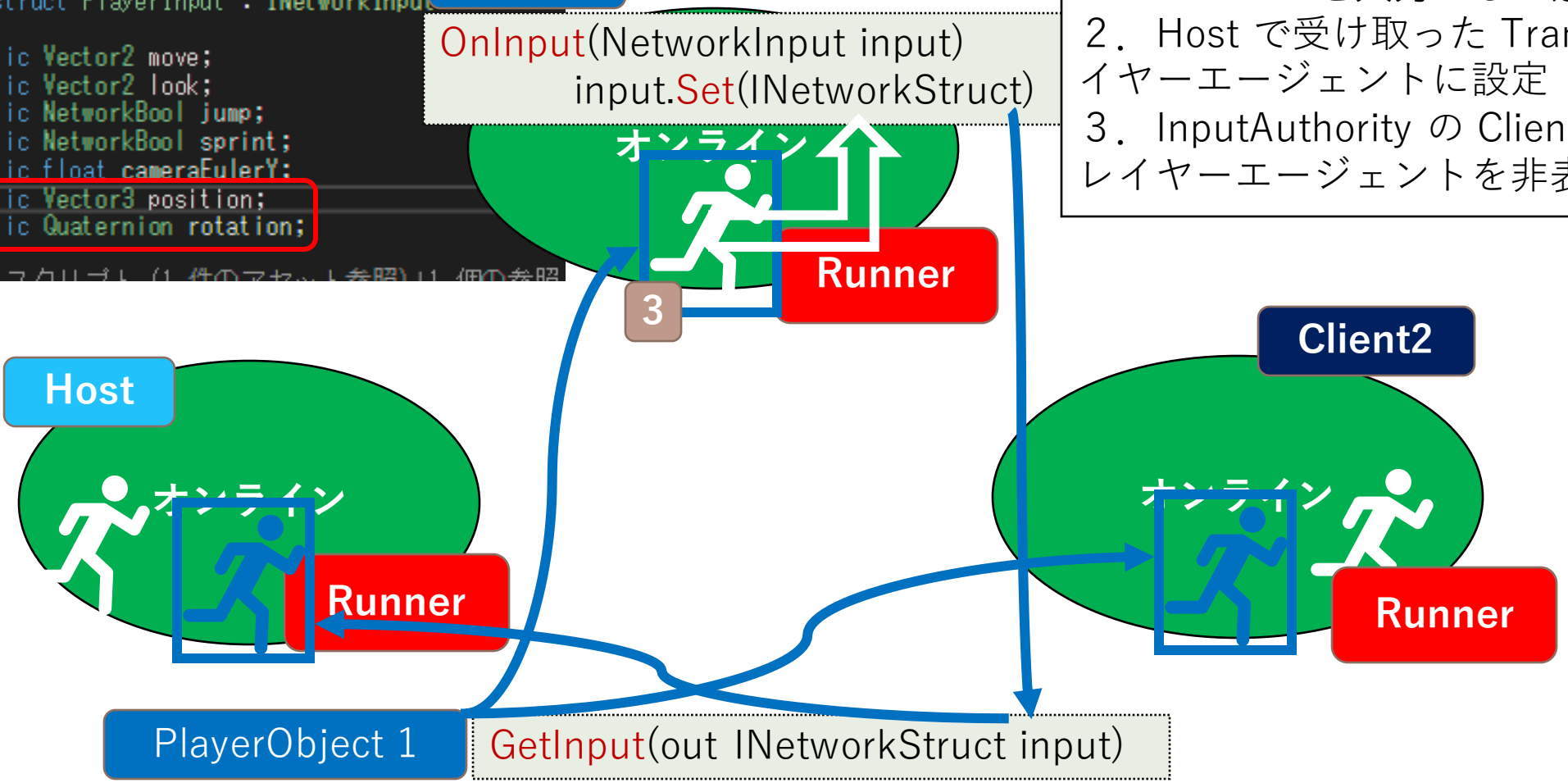
```
3 個の参照
public struct PlayerInput : INetworkInput
{
    public Vector2 move;
    public Vector2 look;
    public NetworkBool jump;
    public NetworkBool sprint;
    public float cameraEulerY;
    public Vector3 position;
    public Quaternion rotation;
}
```

1

Client1

OnInput(NetworkInput input)
input.Set(INetworkStruct)

1．オフラインのプレイヤーエージェントのTransform を入力として送信
2．Host で受け取った Transform をプレイヤーエージェントに設定
3．InputAuthority の Client シーンではプレイヤーエージェントを非表示



2

```
this.starterAssetsInputs.SprintInput(input.sprint);
this.transform.position = input.position;
this.transform.rotation = input.rotation;
}
```

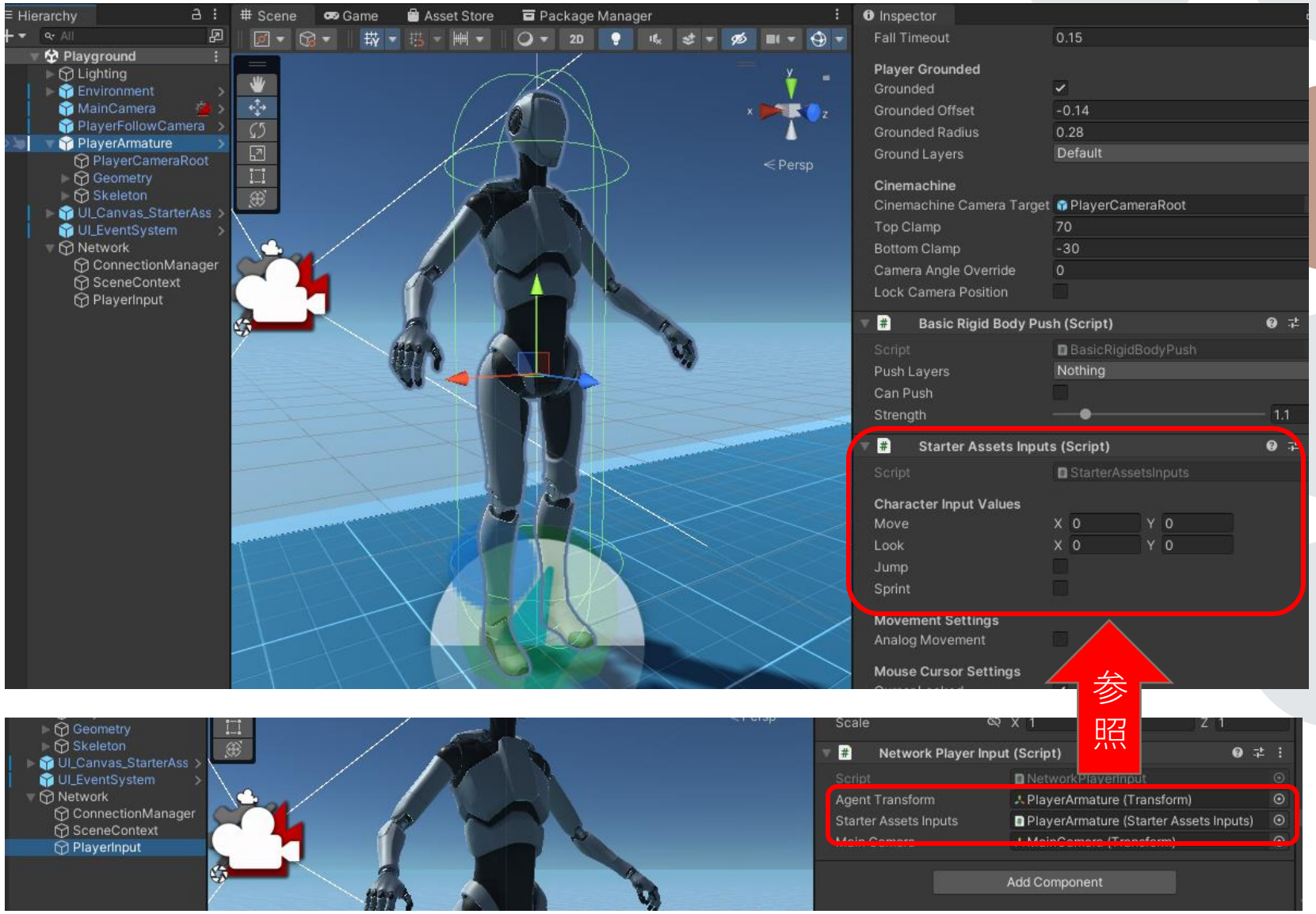
オフラインゲームを非破壊で参照

Host に Client の入力を送信する方法を解説します

オフラインのゲームに一切変更を加えない制約を守っています

ゲーム入力をシーン参照することで解決してます

シーンに配置されている PlayerArmature オブジェクトの StarterAssetsInputs を参照することで、ユーザーの Move, Look Vector2 や Jump, Sprint の状態を参照できます
PlayerAmature の Transform も同様に参照します



Client 入力はどこからでも送信できる

INetworkRunnerCallbacks を実装したクラスを NetworkRunner の AddCallbacks に登録します赤い矢印

OnInput が Tick ごとに呼ばれるようになるため、ここでオフラインのゲーム状態を Set します

Host 側のプレイヤーエージェントの**状態を入力として送信し Host で上書きしてしまえば、Client が状態を持つオンラインゲームもデザインできます**

補足：入力に状態を渡すのは、データ通信量が増えたり、チートに弱くなったりするので、Host ではなくShared モードも検討しましょう（今回は Host 利用にこだわってみました）

NetworkGame クラスの Spawned でインタフェース実装を登録

```
        public override void Spawned()
        {
            this.name = "[Network]Game";
            NetworkSceneContext.Instance.Game = this;
            Runner.AddCallbacks(NetworkSceneContext.Instance.PlayerInput);
        }

        0 個の参照
```

Singleton パターンなら NetworkObject の処理にもシーンの参照を渡せます

```
    3 個の参照
    public struct PlayerInput : INetworkInput
    {
        public Vector2 move;
        public Vector2 look;
        public NetworkBool jump;
        public NetworkBool sprint;
        public float cameraEulerY;
        public Vector3 position;
        public Quaternion rotation;
    }
    Unity スクリプト (1 件のアセット参照) | 1 個の参照
    public class NetworkPlayerInput : MonoBehaviour, INetworkRunnerCallbacks
    {
        [SerializeField, Tooltip("Local pl
        internal Transform agentTransform;
        [SerializeField, Tooltip("StarterAssetsInputs")]
        StarterAssetsInputs starterAssetsInputs;
        [SerializeField, Tooltip("MainCamera Transform")]
        Transform mainCamera;

        0 個の参照
        void INetworkRunnerCallbacks.OnInput(NetworkRunner runner, NetworkInput input)
        {
            input.Set(new PlayerInput
            {
                move = this.starterAssetsInputs.move,
                look = this.starterAssetsInputs.look,
                jump = this.starterAssetsInputs.jump,
                sprint = this.starterAssetsInputs.sprint,
                cameraEulerY = this.mainCamera.eulerAngles.y,
                position = this.agentTransform.position,
                rotation = this.agentTransform.rotation,
            });
        }
```

オフラインゲームの状態を参照

状態を入力として送信

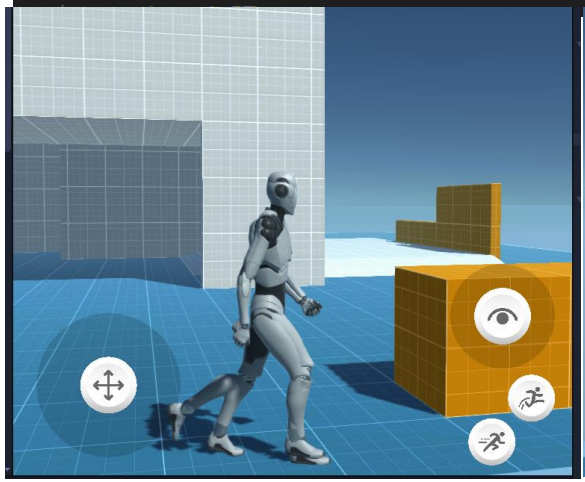
プレイヤーエージェントの Spawn と非表示

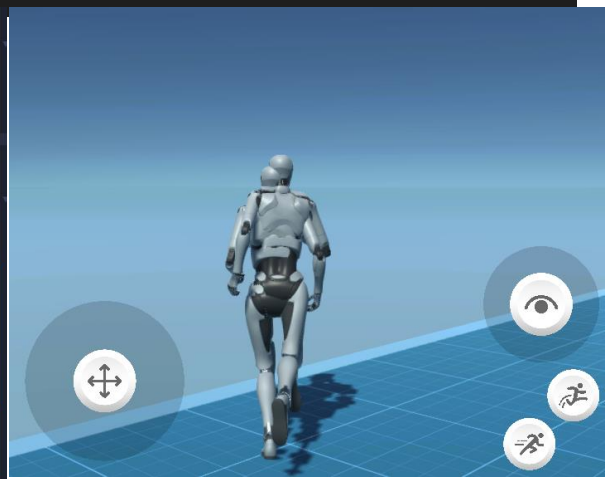
プレイヤーオブジェクトとプレイヤーエージェントは分けて Prefab を用意します

今回は NetworkGame の Join 関数に Spawn するコードを書いています

オフラインゲームで存在しているキャラクターモデルとプレイヤーエージェントが重なって表示されてしまいますので InputAuthority を持っているプレイヤーエージェントの Renderer を非表示にしています

```
public class NetworkGame : NetworkBehaviour
{
    [Networked, HideInInspector, Capacity(200)]
    4 個の参照
    public NetworkDictionary<PlayerRef, NetworkPlayer> Players { get; }

    1 個の参照
    public void Join(NetworkPlayer player)
    {
        if (!HasStateAuthority)
        {
            return;
        }
        var playerRef = player.Object.InputAuthority;
        if (this.Players.ContainsKey(playerRef))
        {
            Debug.LogError($"Player {playerRef} already joined");
            return;
        }
        this.Players.Add(playerRef, player);
        this.SpawnPlayerAgent(player);
    }
```





プレイヤーエージェントのアニメーション同期

プレイヤーエージェントの Owner がプレイヤーオブジェクトを指しています
そのプレイヤーオブジェクトの StarterAssetsInputs の入力をプレイヤーエージェントの SatarterAssetsInputs に設定しています

Transform の同期だけだと、エージェントが棒立ちしたまま移動と回転するだけなので、自然に振る舞わせるようゲームの入力も一緒に送信します（本来の使い方）

Host で取得した入力を一度 State に設定し、他の Client はその State を入力に使ってプレイヤーエージェントのアニメーションを再生します

```
0 個の参照
public override sealed void FixedUpdateNetwork()
{
    if (!IsProxy)
    {
        return;
    }
    if (null == this.starterAssetsInputs || null == this.Owner)
    {
        return;
    }
    Vector2 move = this.Owner.StarterAssetsInputs.move;
    if (null != this.mainCamera)
    {
        move = Quaternion.Euler(0, 0, -this.Owner.StarterAssetsInputs.cameraEulerY + this.mainCamera.transform.rotation.eulerAngles.y) * move;
    }
    this.starterAssetsInputs.MoveInput(move);
    this.starterAssetsInputs.LookInput(this.Owner.StarterAssetsInputs.look);
    this.starterAssetsInputs.JumpInput(this.Owner.StarterAssetsInputs.jump);
    this.starterAssetsInputs.SprintInput(this.Owner.StarterAssetsInputs.sprint);
}
```

それぞれのプレイヤーのカメラの向きも送信させました、このカメラ向きを使えば入力の軸の向きを調整することができ、他プレイヤーの入力もオフラインゲームのコードを修正することなく適用することができました

Transform のなめらかなずれ修正

入力だけを使って動かすと、なめらかに他プレイヤーエージェントが動きました

ただしこれには、積み重なる位置ずれを修正しなければならない課題があります

NetworkTransform を使うと解決するのですが、ガタついてしまったので、なめらかにゆっくりとずれを修正する仕組みがほしいところで、今回作りました

技術ポイント：NetworkBehaviour の Render 関数は MonoBehaviour の Update 関数の代替で利用しよう
（Spawn 後、シミュレーション後になってから呼ばれる利点がある）

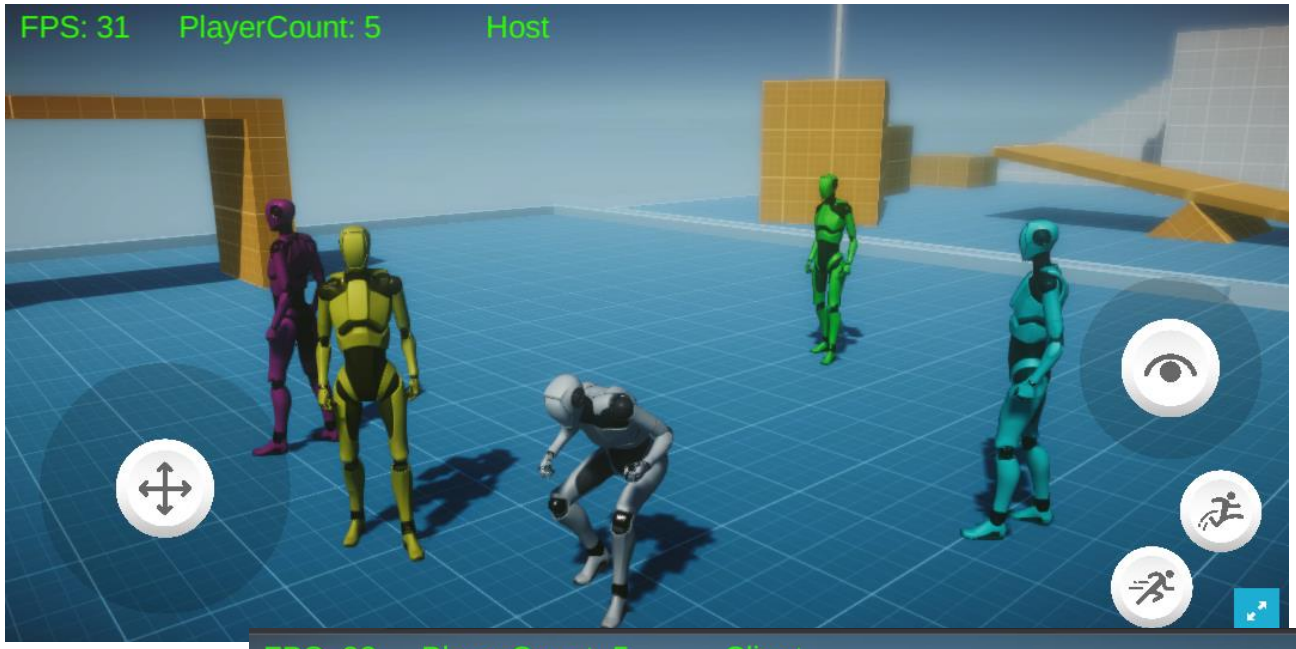
```
public class NetworkSmoothTransform : NetworkBehaviour
{
    [SerializeField, Range(1f, 100f)]
    float smooth = 10f;
    [SerializeField, Range(0.01f
    float warpDistance = 0.5f;

    [Networked, HideInInspector]
    4 個の参照
    public Vector3 position { get; set; }
    [Networked, HideInInspector]
    3 個の参照
    public Quaternion rotation { get; set;

    0 個の参照
    public override void Render()
    {
        base.Render();
        if (HasStateAuthority)
        {
            this.position = this.transform.position;
            this.rotation = this.transform.rotation;
        }
        else if(IsProxy)
        {
            if (this.warpDistance < Vector3.Distance(this.position, this.transform.position))
            {
                this.transform.position = this.position;
                this.transform.rotation = this.rotation;
            }
            this.transform.position = Vector3.SmoothDamp(this.transform.position, this.position, ref th
            this.transform.rotation = Quaternion.Lerp(this.transform.rotation, this.rotation, smooth *
        }
    }

    Vector3 currentVelocity;
}
```

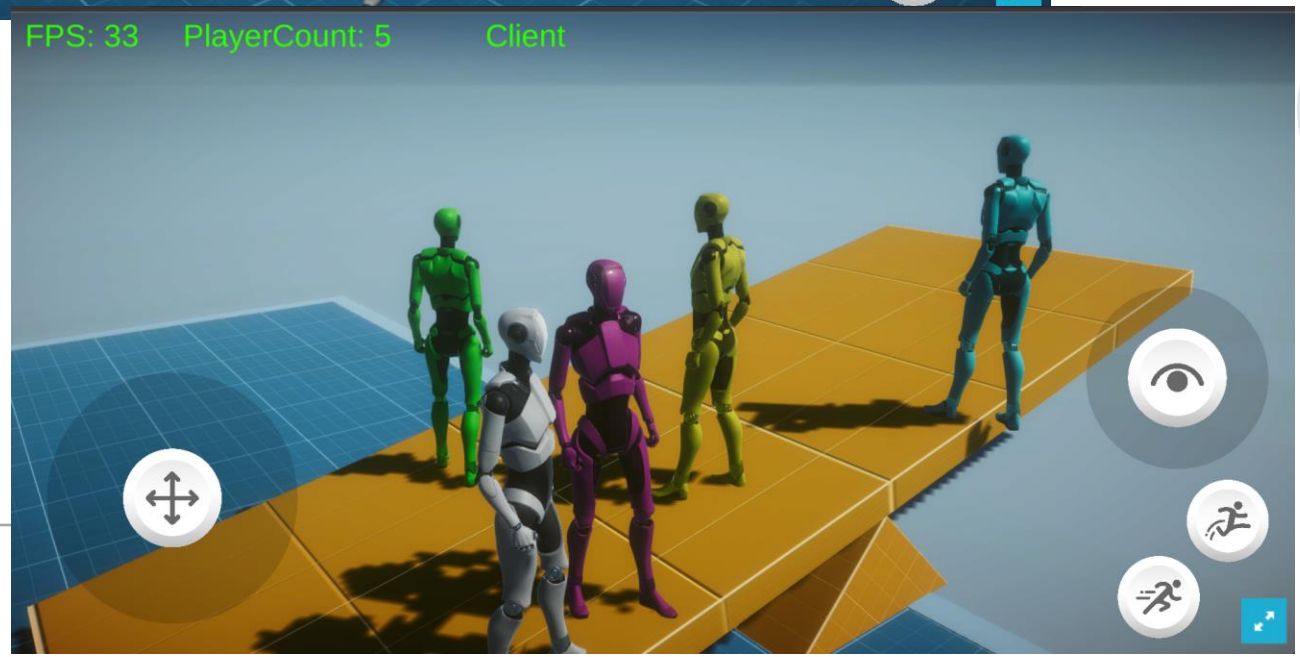
NetworkBehaviour を継承して [Networked] 属性を プロパティに与えて State を同期します

StateAuthority だったら Transform を State に

Proxy だったら State を Transform に設定する

オフラインゲームのオンライン化まとめ





オフラインゲームに変更を加えることなく、他プレイヤーのエージェントがゲームシーンに登場する実装を具体的な手順と Fusion の仕組みの図解を通して解説しました

ぜひみなさんのオフラインゲームをオンライン化してみてください

サンプルプロジェクトはこちらに公開しました→ URL

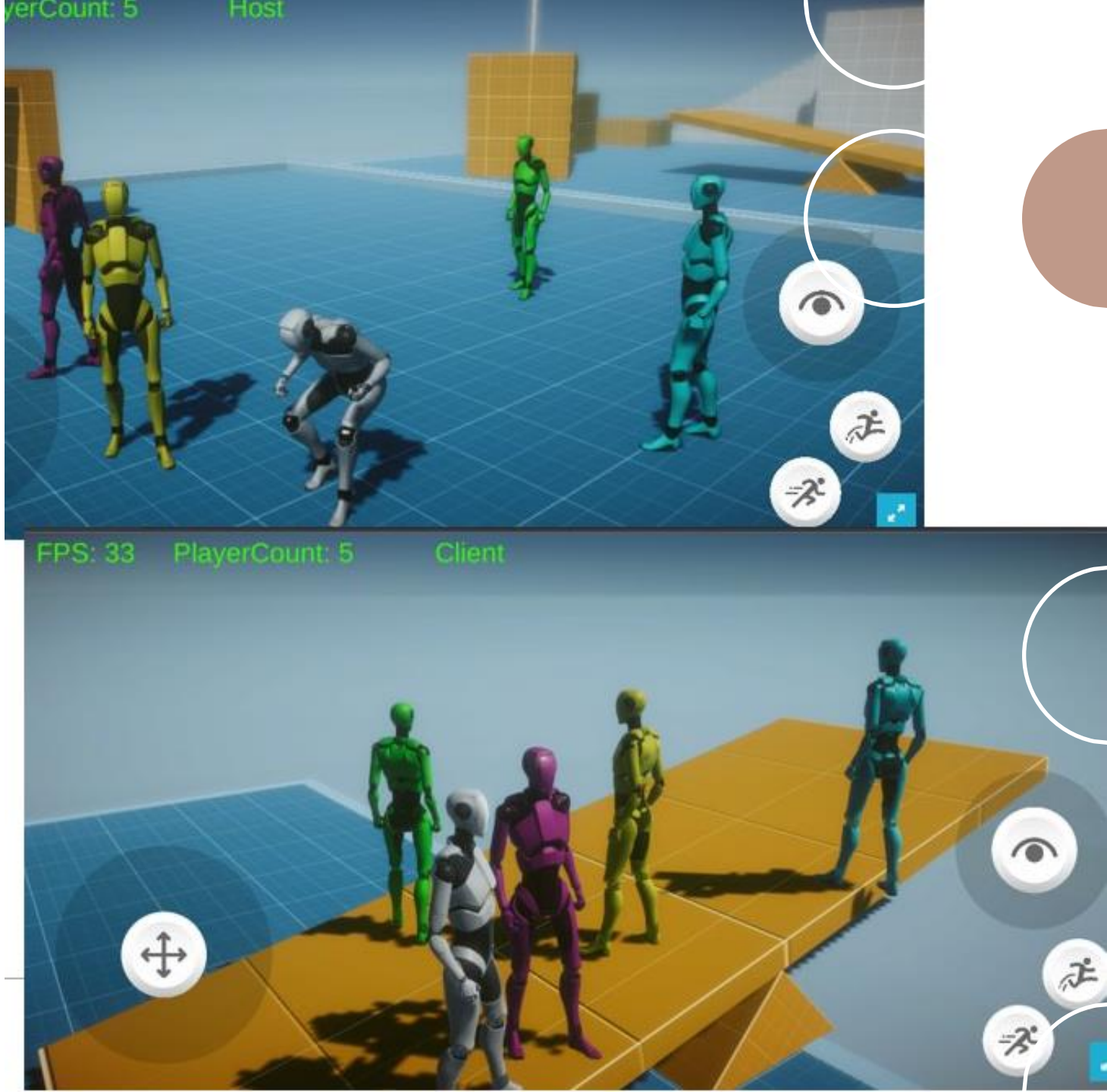
小休止：みんなで遊んでみよう

https://is.gd/Fq0RN3

上記アドレスにアクセスしてみてください

ブラウザでゲームが起動し、すぐにオンライン化して他プレイヤーのエージェントと空間を共有できると思います

公開用に、ちょっとだけゲームをいじりまして、シーソーやボールを配置しています



WebGLビルドのゲームをWeb公開する手順
(用語説明～)

ビルド成果物を Web ページとして公開しても、残念ながらエラーが出て Unity ゲームは動きません

この問題の解決方法を紹介します

まず WebGL ビルド結果のフォルダを AWS の CloudFront のオリジンに設定した S3 に配置します…

少し用語説明からしますね

AWS とは **Amazon Web Services**

有名なクラウドサービスですね

CloudFront は AWS のサービスの一つでデータ配信に使えます

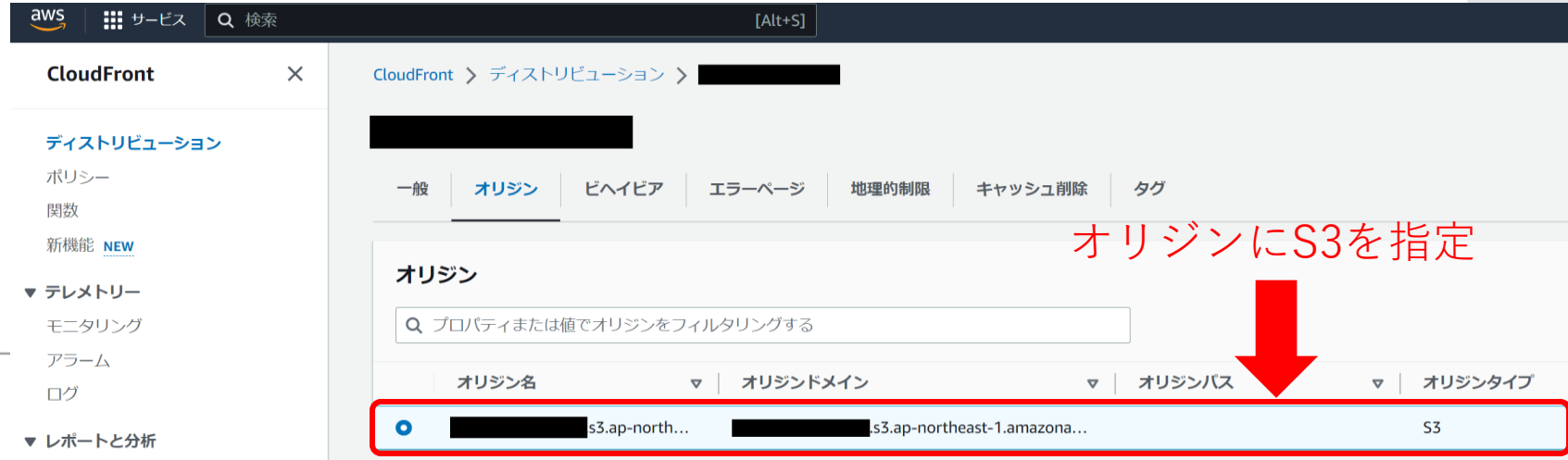
S3 は AWS のサービスの一つでデータ配置に使えます

CloudFront で配信するデータの一番元となるデータ置き場に S3 を設定することができて、これをオリジンに設定したと表現しました

つまり… CloudFront のオリジンに設定した S3 にデータを置くと、URL など Web ブラウザ経由でデータを配信できるようになるわけです

じゃあ WebGL ビルド結果のフォルダをその CloudFront のオリジンに設定済みの S3 に配置すればゲーム公開完了？

というところで、そうもいかないエラーが発生するので、その課題を対象にこれから説明をはじめます



メタデータを .gz ファイルに追加

Build フォルダ以下に .gz 拡張子のファイルが 3つあります

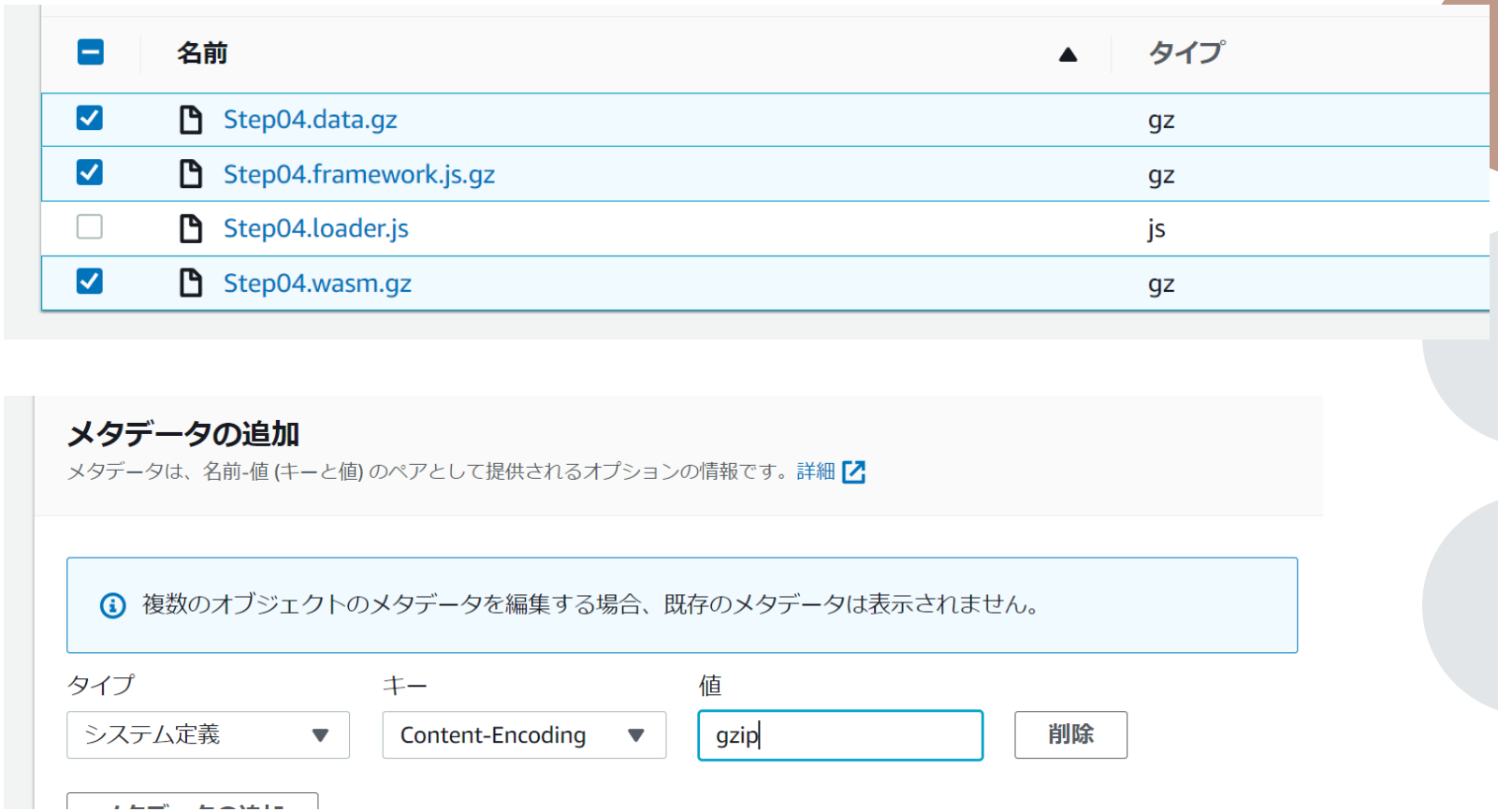
これらのファイルにメタデータの追加を行い

タイプ：システム定義

キー：Content-Encoding

値：gzip

を設定します



メタデータを .gz ファイルに追加2

.wasm.gz ファイルだけ追加でメタデータの追加を行います

タイプ：システム定義

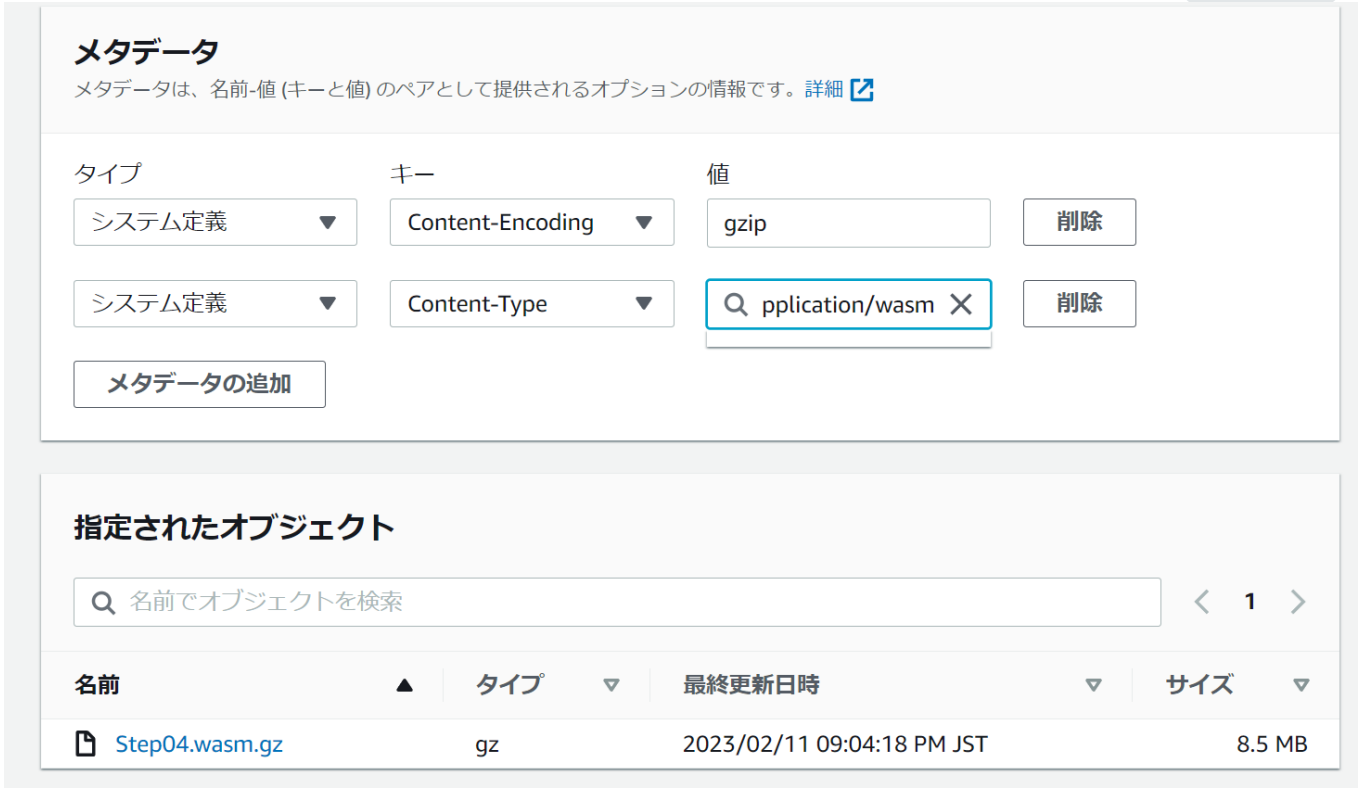
キー：Content-Type

値：application/wasm

を設定します

以上です

このメタデータ追加によって、エラーが解消されます



参考にした公式サンプル

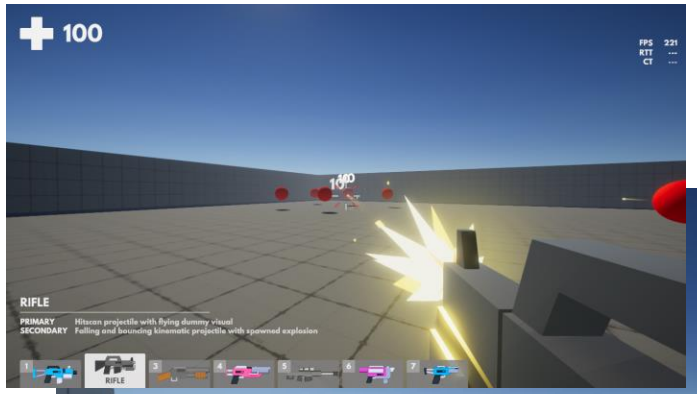
Fusion Projectiles: URL

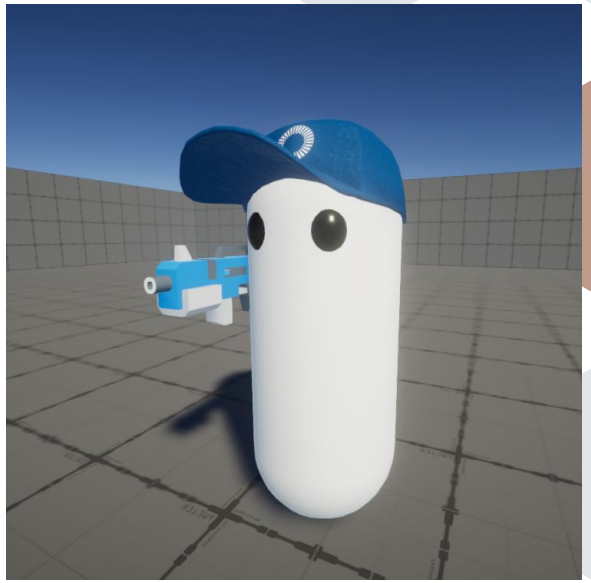
- 複数人でシューティングゲームするサンプル
- シーンに登場する多数の弾丸をどうやって同期すると効率的なのか回答が示されています

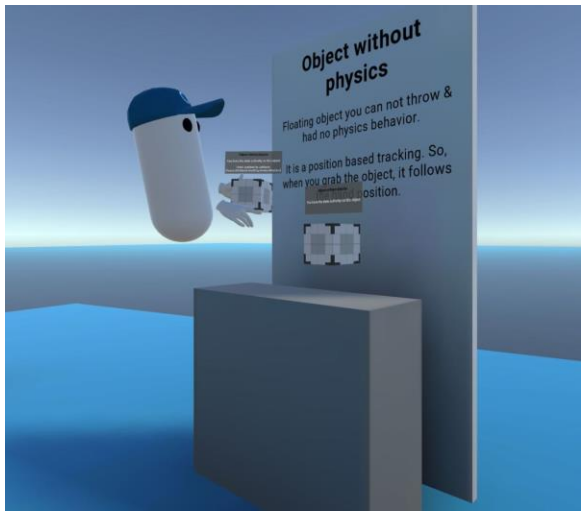
Fusion VR ホスト: URL

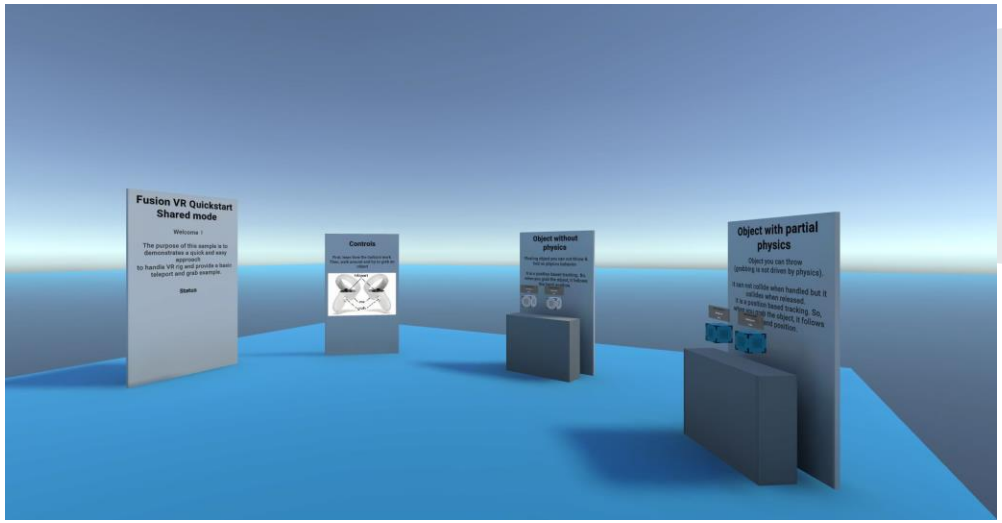
Fusion VR 共有: URL

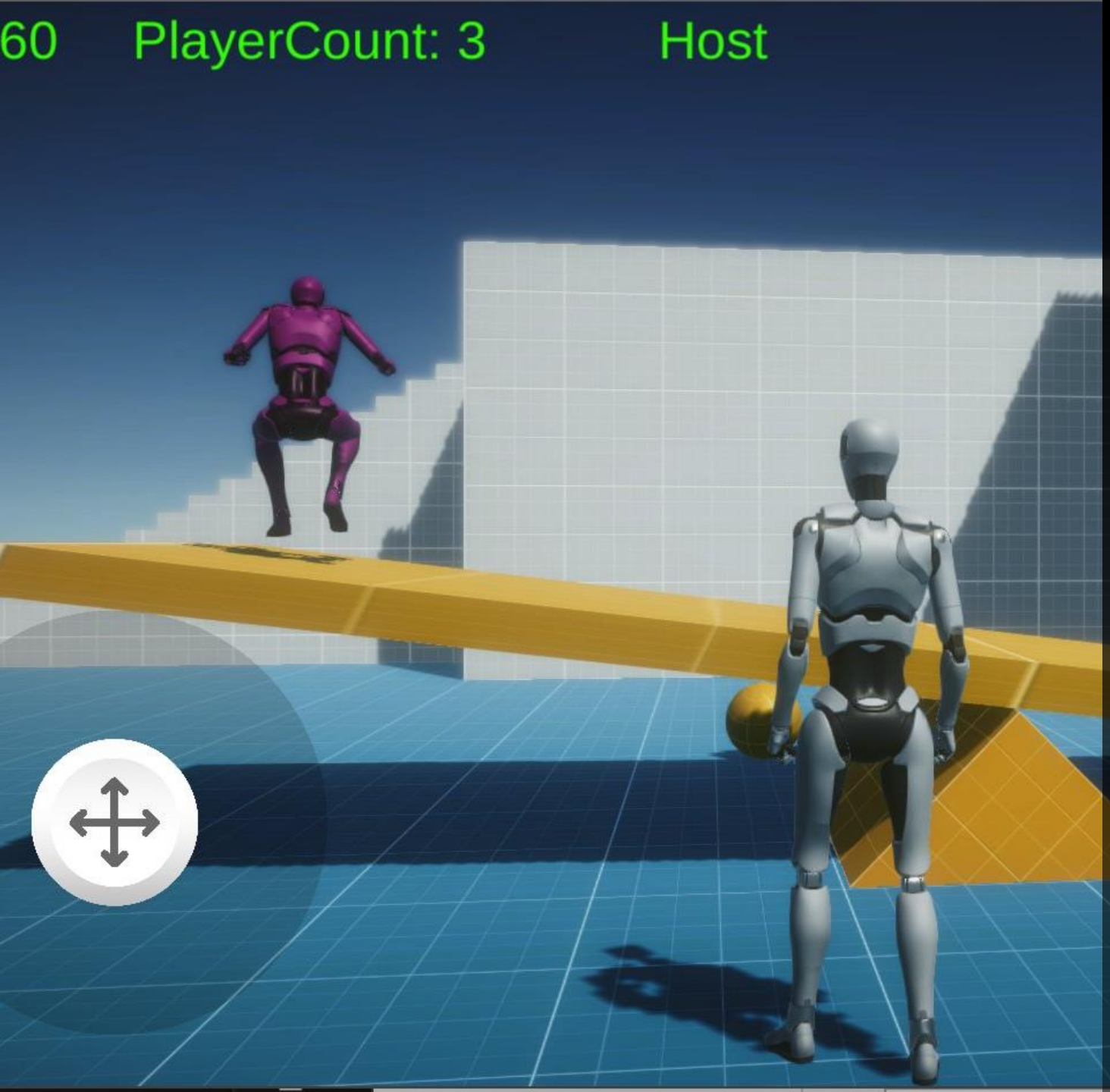
- VR機器をつけて複数人でワールドを共有し、シーンのモノをつかんだり投げたりするサンプル











発表は以上です
ご清聴ありがとうございました