ネットワーク・ゲーム開発

1. ゲームでのネットワーク運用形態

一般的にゲームのネットワークでは TCP/IP を利用しており、現在個別認証には IPv4 が利用されている。

その IPv4 を利用したゲームで利用するネットワークシステムの運用形態をまず考えてみる。運用形態には大きく分けて下記の二つがある。

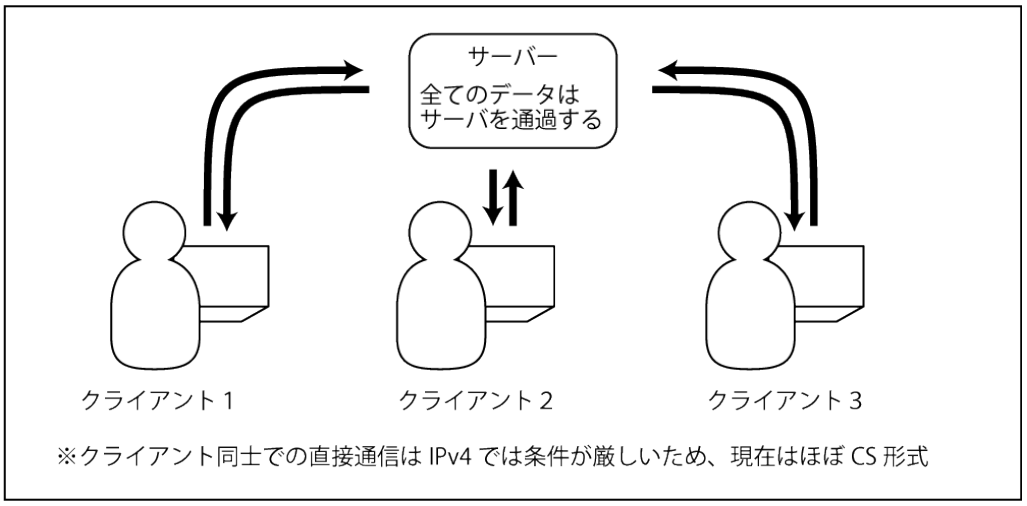
○ CS(クライアント・サーバシステム)システム
○ P2P(ピア・トゥー・ピア)システム

それぞれの特徴は下記の通り

1.1. CS システムの特徴

システム全体を統括しサービスを提供する “サーバ” と、サービスの提供を受ける “クライアント”に分けて構成する。

イメージ図



運用環境として、“クライアント” だけでなく “サーバ” を同時に開発する必要があるため、開発の難易度は多少高い。

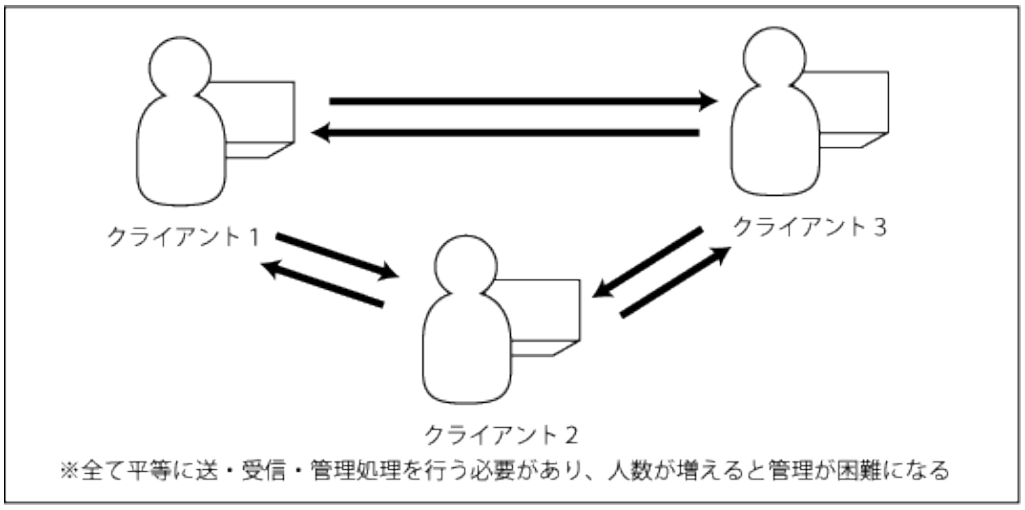
マルチユーザで運用する場合、ユーザ管理など DB などを利用してサーバで管理できるため、クライアント側にあまり負荷をかけずに大規模運用が可能になる。

基本的に全てのデータが、サーバを経由するため、ユーザ・アイテムの管理、チートなどの監視など様々な機能を実装できる。

1.2. P2P(ピア・トゥー・ピア)システムの特徴

ホストとピアのプログラムを同一のプロジェクトで開発できるので、比較的楽に実装できる。

イメージ図



全体管理ができないため、機能の追加時の配布などが困難であり、また人数が増えると処理負荷の増加や通信管理が困難になる。また IP 変換を利用している環境では、直接通信ができない。

また、個別のクライアントで不正が行われても防ぐことが非常に困難になる。

1.3. 通信プロトコルの選択

TCP/IP で通信を行う場合、通信品質と運用環境を考慮し、TCP/UDP の選択が必要になる。それぞれ特徴があるため、適切なプロトコルの選択が必要になる。

○TCP

プロトコルに再送などのエラー処理が組み込まれているため、受信データの品質が高いが、思わぬ遅延が発生しやすいためリアルタイムでの処理には適さない。リアルタイム性が必要のない、ボードゲーム、チャットなどに適している。

○UDP

動画や音楽のストリーミングなど、リアルタイム性が必要な用途に利用される。エラー処理などクライアント側での処理が必要になるが、通信のタイミングをコントロールしやすい。

1.4. 通信のシリアライズ

TCP/IP 通信のパケットは、全て Byte データでやりとりされる。そのため送受信データの形式を Byte データに変換及び、復号する必要がある。これをシリアライズという。

また、通信する Byte データの並びは CPU のアーキテクチャによって異なる場合があるため「ネットワークバイトオーダ」のルールがあり、ビッグエンディアンが用いられる。

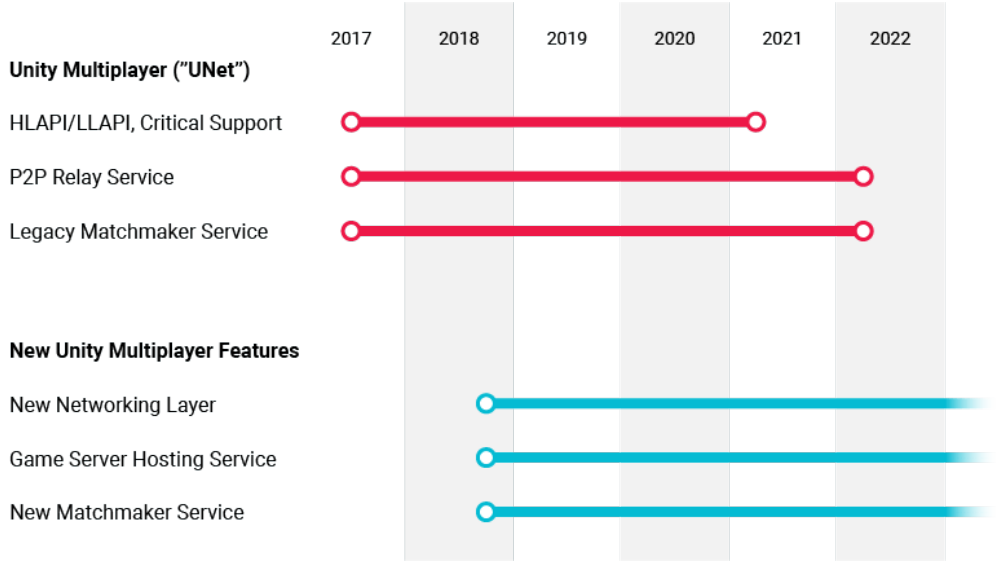
2. Unity のネットワークゲームフレームワーク UNET

前節の通り、ネットワークを利用するために、多くの実装項目がある。基本的な内容はゲームによって同様の内容も多いが、ゲームの安定運用やネットワークパフォーマンスの追及を行うため独自の実装を行う必要があり、ハードルが高い。

そこで Unity には、ネットワークを利用したゲーム開発を行うための環境が用意されている。それが UNET で Unity2017 より実装されている。

2.1. 次世代の Unity ネットワークゲームフレームワーク

Unet は Version5 から実装され、現在、次世代の環境への移行を計画している。



2018 より徐々にその機能の提供が切り替わることになるが、ネットワークゲームの基本機能を学習する上で Unet で主たる機能の学習は可能であり、今回 Unet の環境を使いネットワークゲームへの実装制作を通して、ネットワークゲームで必要な技術を学習する。

3. Unet を用いたネットワークゲーム開発

Unet を利用したネットワーク対戦シューティングゲームの制作を行い必要な機能について学習する。

3.1. Netwok ゲームの制作

本来 Network ゲームの開発では、Network に関する要素を考慮して作成する必要があるが、Unet では、最初から高機能なライブラリが用意されているため、比較的容易に Network ゲームの基礎を体験できる。

今回通常のクライアント作成から、Unet による Netowrk 要素を徐々に付加する形式でゲーム作成を行いながら、Network ゲームで必要な技術の習得を目指す。

3.2. NetworkManager と NetworkManagerHUD 及び NetworkIdentity

Unet を用いてネットワークゲームを作成する際、キーになるコンポーネントがある。順を追ってそのコンポーネントの説明を行う。

3.2.1. NetworkManager

Unet の NetworkManager コンポーネントは、ゲームのネットワークの状態を管理する。このコンポーネントにはサーバ機能も含まれるため CS システムが容易に実装できる。

サーバ機能については、クライアント間でのデータの通信管理だけでは無く、ゲーム状態の管理、オブジェクト生成（Spawning）の管理、シーン管理、デバッグ情報、マッチメイキングなど実装されている機能は多岐にわたる。今回、簡易な CS システム実装を行うが、ネットワーク内の 1 台のみサーバ兼クライアントの役割を担うことになる。

3.2.2. NetworkManagerHUD

NetworkManagerHUD を用いることで、ソフトウェア起動時に、上記 NetworkManager の動作環境を、下記のように設定するための UI を提供する。

○NetworkManager.StartClient　：クライアント機能を開始する。
○NetworkManager.StartServer　：サーバ機能を開始する。
○NetworkManager.StartHost　：同一アプリ内でクライアント、サーバ両機能を開始する。

3.2.3. NetworkIdentity

Network ゲームは、各クライアントで同一のクライアントソフトが動作することになる。そのため、各クライアントで自分が利用しているクライアントで制御するオブジェクトと、別クライアントからの情報により動作するオブジェクトの判別を行う必要がある。それを識別するために NetworkIdentity コンポーネントが準備されている。

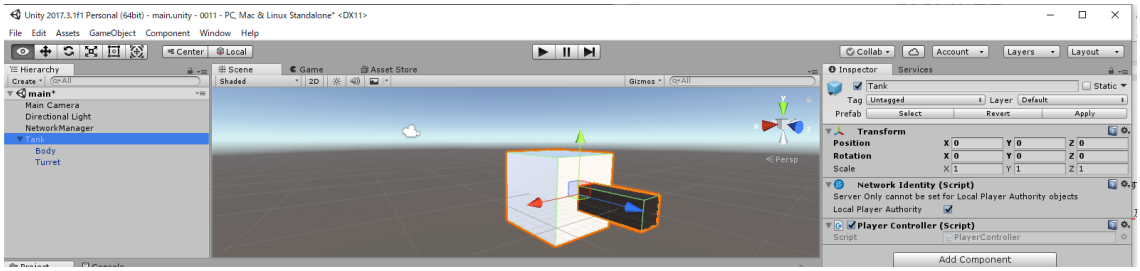
NetworkIdentity を Local Player Authority に設定すると、自分が利用しているクライアントで制御している、オブジェクトの動きを制御できるようになる。

4. ゲームの実装

4.1. Player Prefab の作成

NetworkManager によって、各シーン上に Player を生成できる。まずは、そのための prefab オブジェクトを準備する。各自適当な Prefab を準備しておく。ここでは例として簡易な下記のモデルを使用することとする。

○空のゲームオブジェクトを(0,0,0)に作成し、名前を “Tank” にする。
○その下に Cube を作成し、名前を “body” に変える。
○“Tank”下に Cube を作成し、名前を ”Turret” に変更する。
○“Turret” の位置、スケールを Position(0,0,0.5) , Scale(0.2,0.2,1.5) に変更する。
○新規マテリアル“Turret” を作成し、色を黒(0,0,0) に変更した後、先ほどの ”Turret” に適用する



これを Player として、Player がキーボードで動けるよう Script：PlayerController.cs を作成後、Player にアタッチし動作することを確認し、 動作確認後、Player を Prefab 化しておく。

PlayerController.cs

```
1   using UnityEngine;
2
3   public class PlayerController : MonoBehaviour
4   {
5       public float RoteteRate      = 150.0f;
6       public float RateRate     = 3.0f;
7       void Update()
8       {
9           var x = Input.GetAxis("Horizontal") * Time.deltaTime * SpeedRate;
10          var z = Input.GetAxis("Vertical") * Time.deltaTime * RotateRate;
11
12          transform.Rotate(0, x, 0);
13          transform.Translate(0, 0, z);
14      }
15  }
```

4.2. Player のネットワーク対応

現在スタンドアロンで、画面上に配置したオブジェクトがキーボードで動くだけのものだが、これをネットワーク化する。そのためには、3.2 にある各コンポーネントを、利用する事になる。

4.2.1. NetworkManager コンポーネントの実装

ゲームシーン内に NetworkManager を配置する。そのためには、空のオブジェクトを Hierarchy ビューに配置し空のオブジェクトの名前を “NetworkController” に変更した後、

Component ＞ Network ＞ NetworkManager
Component ＞ Network ＞ NetworkManagerHUD

にて各コンポーネントをアタッチする。その後、Unity エディタの Play ボタンを押し、実行する。そうすると NetworkManagerHUD により下記のような UI が表示されるので、赤枠で囲んだ “LAN Host” ボタンを押すと、クライアント･サーバの機能を持った状態で起動する。そうすると先ほどと同じように動作するが、まだこのままでは Network に対応できていない。

一旦停止して、Network 上で認識できるように準備した Prefab か、先ほど作成した“Tank” に NetworkIdentity コンポーネントをアタッチする。

Compornent ＞ Network ＞ NetworkIdentity

アタッチ後、Inspector で下記の通り ”Local Player Authority” にチェックを入れる。
これにより、NetworkManager で生成された Player オブジェクトが各クライアントで制御できるようになる。

しかし、実行すると Console に下記のエラー表示がされているはずだ。

```
The PlayerPrefab is empty on the NetworkManager. Please setup a PlayerPrefab object.
UnityEngine.Networking.NetworkIdentity:UNetStaticUpdate()
```

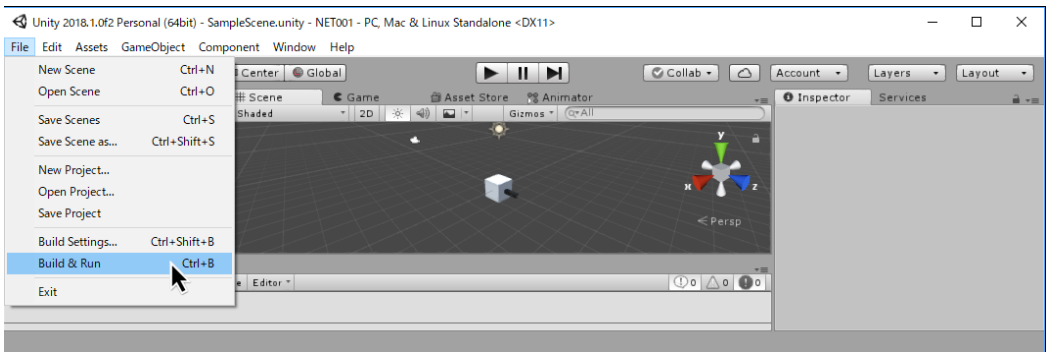
このエラーは、NetworkManager が生成すべき Prefab が設定されていないために起こる。現在、シーン上に Player オブジェクトを配置しているが、あくまでクライアント上に独立しているオブジェクトであり、他のクライアントと接続した場合表示されない。実際確かめてみよう。

4.2.2. Network ゲームの動作確認

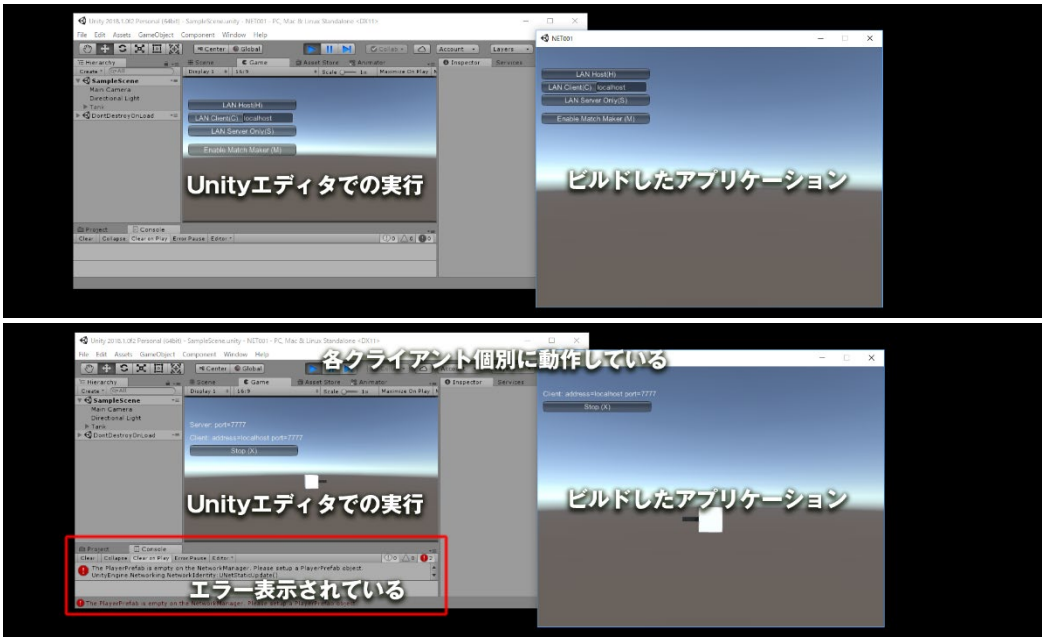
Network ゲームの動作確認を行うためには、複数のクライアントを起動する必要がある。この場合、作業マシンと別に動作環境を準備できればいいのだが、その環境が準備されていない場合も多い。

その場合、1 台の開発マシンで Network ゲームの動作確認するために環境は TCP/IP で準備されている。TCP/IP には特別な IP アドレスとして “localhost(127.0.0.1)” が準備されており、自分自身に向け IP 通信を行い Network 通信が可能である。

ふたつの動作環境を作成するために、Unity エディタより File ＞ Build&Run を行い現在のソフトを単体アプリとして立ち上げる。このアプリと Unity エディタ側の実行環境で通信確認を行うことで、実際の Network 上での動作確認を行うことが可能になる。

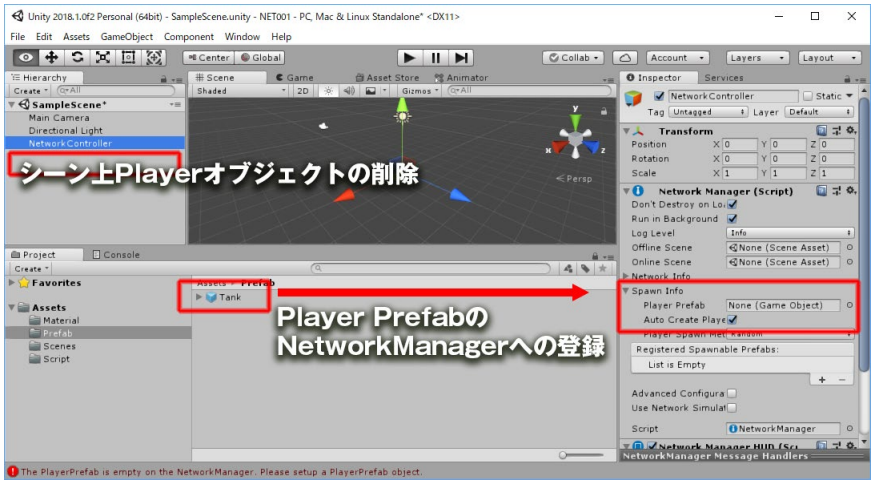


起動した後、一方を “LAN HOST”、もう一方を “LAN Client” で起動することで、2 つのソフトが Network で接続される。しかし、各々の画面に表示されるオブジェクトは 1 つで、各画面でしか動作しない。



4.2.3. Player Prefab の NetworkManager への登録

いったん停止し、今度は先ほど作成した Player Prefab を NetworkManager に登録する。登録することで、NetwrokManager が、各クライアントに対して、Player オブジェクトを生成し各々制御できるようになる。



NetworkManager により Player を生成するため、上図のように、シーン上にある Player オブジェクトを削除し、当該 Player の Prefab を NetworkManager に登録する。

Player オブジェクトは生成されたが、まだ個別に動作している。

次は、このオブジェクトの動きを Network 化していく。そのためには、スクリプトの Network 対応が必要になる。次節でその方法を解説する。

4.2.4. Player のネットワーク化

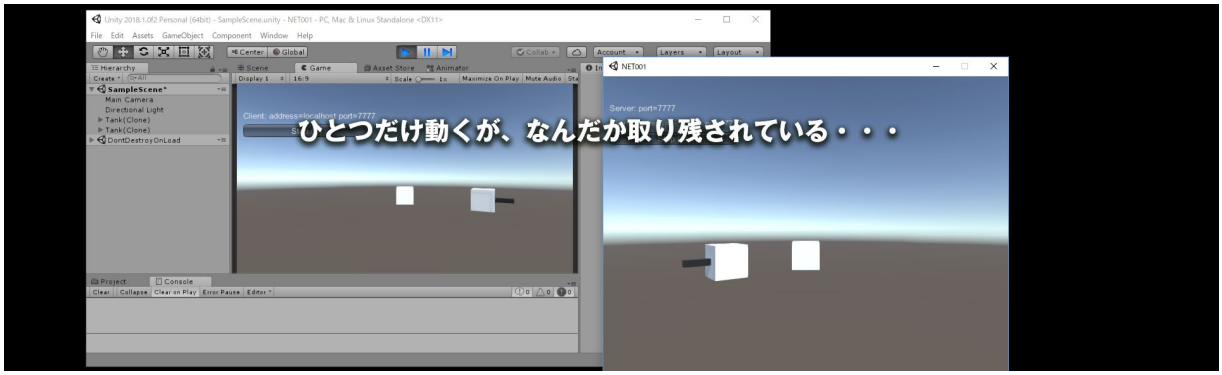
プレイヤーの動きをネットワークと連携させ、ローカルプレイヤーのみがローカルプレイヤーゲームオブジェクトを制御できるように、PlayerController スクリプトを変更する。基本的に、Player オブジェクトは各クライアントで生成されるが、中身が同じであるため、ローカルプレイヤーである場合のみキー入力処理を行うようにスクリプトを変更する。

まず PlayerController スクリプトへ、名前空間 UnityEngine.Networking を追加し、基底クラスを MonoBehaviour から NetworkBehaviour に変更させる。

その後、Update 関数内で、ローカル Player オブジェクトを判別するために、NetworkBehavior にて用意されている "isLocalPlayer" を用いて、ローカルプレイヤー以外はキー入力を受け付けない処理を付加する。

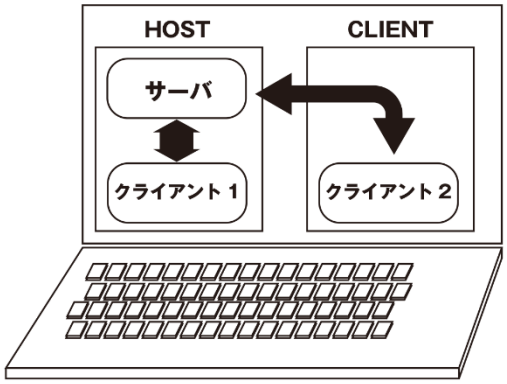
PlayerController.cs　Network 対応

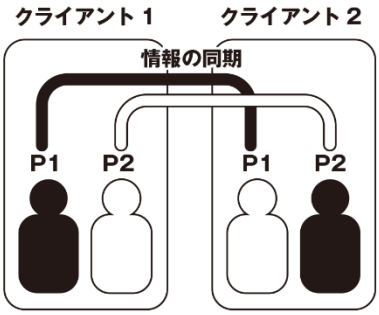
```
1   using UnityEngine;
2   using UnityEngine.Networking;          // Network 用  namespace 追加
3
4   public class PlayerController : NetworkBehaviour    // 基底クラスの変更
5   {
6       public float RoteteRate      = 150.0f;
7       public float RateRate     = 3.0f;
8
9       void Update()
10      {
11          if( !isLocalPlayer ) return;       // ローカルプレイヤーでなければ以降処理しない
12
13          var x = Input.GetAxis("Horizontal") * Time.deltaTime * SpeedRate;
14          var z = Input.GetAxis("Vertical") * Time.deltaTime * RotateRate;
15
16          transform.Rotate(0, x, 0);
17          transform.Translate(0, 0, z);
18      }
19  }
```



4.2.5. 現在の状況

現時点では、クライアント兼サーバ 1 台、クライアント 1 台の構成になっている。





ふたつのクライアントで、それぞれプレイヤーがいる場合、合計で 4 つのプレイヤーオブジェクトを制御する必要がある。

4.2.6. マルチプレイヤー