1. Uniy で利用できるネットワーク環境の概要

Unity では、ユーザーは 2 種類のネットワーク機能を使うことができる。

○マルチプレイヤーゲームを作成する。これには NetworkManager や 高レベル API を使用する必要あり。

○ネットワークのインフラと高度なマルチプレイヤーゲームを構築できる。これには Transport Layer API を使用する。

1.1. 高レベル API（HLAPI）のスクリプト

Unity には「高レベル API（HLAPI）」がある。この API スクリプトを使用することで「低レベル」な実装について知る必要なく、マルチプレイヤーゲームに必要な、ほとんどの共通要件をカバーするコマンドにアクセスできる。

HLAPI の概要は以下の通り。

○「ネットワークマネジャー（Network Manager）」の機能を使用して、ゲームのネットワーク状態をコントロールする。

○“クライアントホスト” としてゲームを操作する。

○汎用シリアライザー。

○ネットワークメッセージの送受信。

○クライアントからサーバーにネットワークのコマンドを送信。

○サーバーからクライアントにリモートプロシージャコール(RPC)を行う。

○サーバーからクライアントにネットワークイベントを送信。

1.2. 利用環境

Unity のネットワーキングは、ゲームエンジン/エディターに統合されている。マルチプレイヤーゲームを作成するためコンポーネントで制御を行い、視覚的にビルドすることができる。そのために Unity は以下の機能を提供している。

○ネットワーク上にあるオブジェクトの NetworkIdentity コンポーネント

○ネットワーク対応のスクリプト NetworkBehaviour

○Transform オブジェクトの自動同期（automatic synchronization）を設定します。

○スクリプト変数の自動的同期化。

○ネットワーク上のオブジェクトを配置するためのサポート。

○Network components

1.3. 提供されるインターネットサービス

○マッチメイキングサービス

○ゲームマッチの呼びかけ、マッチの作成

○利用および参加可能なマッチメイキングのリストアップ

○リレーサーバー（Relay server）

○非専用サーバーを利用したインターネットを通じてのゲームプレイ。

○マッチ参加プレーヤーに向けてのメッセージ・ルーティング

1.4. NetworkTransport リアルタイムトランスポートレイヤー

Transport Layer API で以下の機能を提供します：

○ゲームに最適化された UDP ベースプロトコル

○Head of Line Blocking（HoLB; 行頭ブロッキング)の問題を回避するためのマルチチャンネル設計

○チャネル当たりネットワーク上で提供するサービス品質(QoS)のさまざまなレベルに対応するサポートの提供

○P2P やクライアント・サーバ・アーキテクチャをサポートする柔軟性のあるネットワークトポロジー

2. 簡易マルチプレイヤーサンプル

Unity ビルトインのマルチプレイヤーネットワーキングと HLAPI を利用することで、マルチプレイヤープロジェクトの開発を効率的に行える。

ここでは、ごく簡単なアセットとスクリプトによって、マルチプレイヤーネットワーキングゲームをゼロからセットアップする。

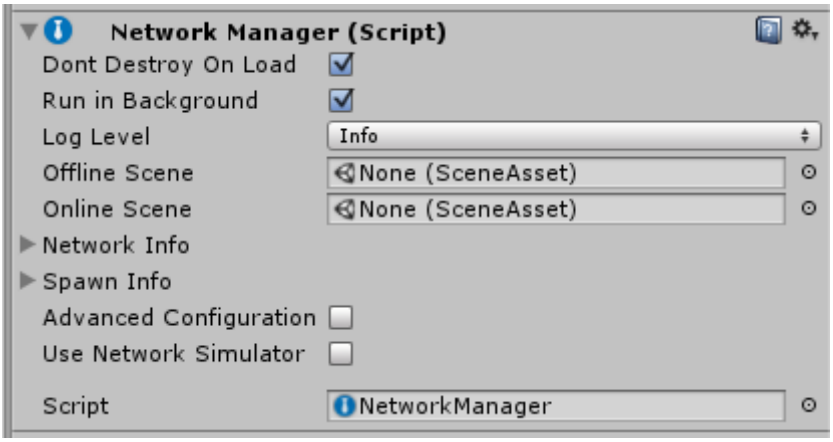
引用：https://unity3d.com/jp/learn/tutorials/topics/multiplayer-networking/introduction-simple-multiplayer-example?playlist=47321

2.1. ネットワークマネージャー（NetworkManager）

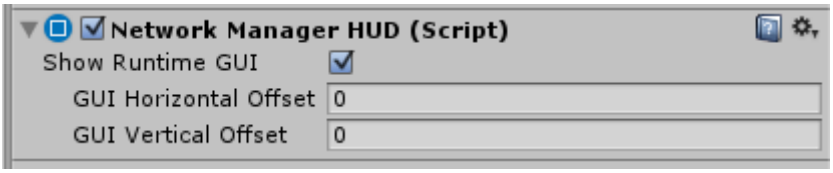
NetworkManager を使って、マルチプレイヤープロジェクトのゲームの状態、Spawn、シーン管理、マッチメイキング、デバッグ情報へのアクセス許可などの管理を行う。そのために、まずプロジェクトに Network Manager を作成する必要がある。

NetworkManager を新規作成するには、新しい空のゲームオブジェクトを作成し、NetworkManager および NetworkManagerHUD コンポーネントを追加する。

NetworkManager コンポーネント



NetworkManagerHUD コンポーネント

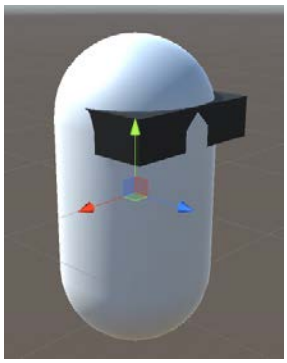


NetworkManagerHUD コンポーネントは NetworkManager と連携して機能し、ゲームの実行中にそのネットワーク状態を制御するための、下記のような簡単なユーザーインターフェースを提供する。

2.2. Player プレハブのセットアップ

初期設定では NetworkManager コンポーネントは、プレイヤープレハブのクローンを作成してそれをゲーム内に生成させることによって、ネットワークゲームに接続する各プレイヤーのゲームオブジェクトをインスタンス化する。

ここでは、プレイヤーの方向が分かるようにするため、下記のオブジェクトを用いる。



このゲームオブジェクトの作成方法は下記の通り、

1. Capsule プリミティブ・ゲームオブジェクトを作成する。
2. 作成したゲームオブジェクトの名前を “Player” に変更する。
3. Cube プリミティブを、Player の子として新規作成する。
4. 作成した新しいゲームオブジェクトの名前を “Visor” に変更する。
5. Visor のスケールを (0.95, 0.25, 0.5) に設定する。
6. Visor ゲームオブジェクトの位置を (0.0, 0.5, 0.24) に設定する
7. マテリアルを新規作成し、名前を “Black” に設定し、黒色の設定する。
8. Visor ゲームオブジェクトのマテリアルを Black にアタッチする。

NetworkIdentity コンポーネントは、ネットワーク上でオブジェクトを識別可能にし、ネットワーキングシステムにそれを認識させるために使用されます。

Player を一意的な Networked GameObject として識別させるために、 NetworkIdentity を Player に追加する。

その後、NetworkIdentity の Local Player Authority のチェックボックスをオンにする。

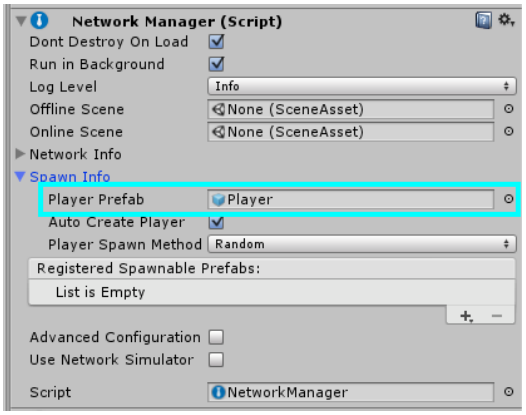
NetworkIdentity を Local Player Authority に設定すると、クライアントがこの Player ゲームオブジェクトの動きを制御できるようになります。

最後に、Player ゲームオブジェクトをプレハブ化し、シーンから Player オブジェクトを削除する。

2.3. プレイヤープレハブを NetworkManager に登録する

Player プレハブを作成後、ネットワーキングシステムに登録する。NetworkManager はこのプレハブを使用して新しい Player ゲームオブジェクトをシーンに生成するが、この Player プレハブは、該当のプレイヤー用に生成するプレハブとして、ネットワークマネージャー に登録されている必要がある。その手順は下記の通り。

○Hierarchy ウィンドウで Network Manager ゲームオブジェクトを選択する。
○NetworkManager のインスペクターで Spawn Info のドロップダウンメニューを開く。
○Player プレハブを Player Prefab フィールドにドラッグする。



NetworkManager コンポーネントは、プレイヤーを含むネットワーク上のオブジェクトの管理に使用される。ほとんどのゲームでは、特定のプレイヤーを単一のプレハブを使用するため、各プレイヤープレハブに対して専用のスロットが NetworkManager に存在する。

プレイヤーを表す新しいゲームオブジェクトは、各プレイヤーのクライアントがホストに参加すると作成される。

2.4. Player の動きを作成する（シングルプレイヤー）

“PlayerController” スクリプトによって、単一 Player ゲームでプレイヤーを動かす。この PlayerController スクリプトは Networking コードを一切使用せずに書かれているため、シングルプレイヤー環境のみで機能する。

“PlayerController” という名前のスクリプトを新規に作成し、Player プレハブにアタッチする。

PlayerController.cs

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

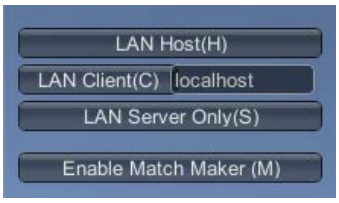
これで、シングル Player での動作が設定できる

2.5. プレイヤーの動きをオンラインでテストする

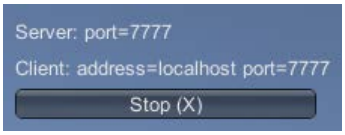
2.5.1. ホスト上でプレイヤーの動きをテストする

この時点では、ネットワークには一切接続されていない状態であるため Player ゲームオブジェクトはクライアント上でしか動作できない。

動作テストを行うには、NetworkManagerHUD のユーザーインターフェースを表示し、ゲーム内 UI の LAN Host ボタンをクリックする。



LAN Host ボタンを押すことで、NetworkManager が、参照先のプレイヤープレハブから新しいプレイヤーゲームオブジェクトをシーン内に作成する。その後、NetworkManagerHUD の表示が変化し、サーバーがアクティブになったことを示す。



このゲームはサーバーであると同時にクライアントでもある、「ホスト」として実行されている。ゲームのホスティングを終了するには、ゲーム内 UI の “Stop” ボタンを選択する。そうすると、ゲームがオフラインモードに戻る。

テストを終了するには、エディターで Play モードを終了させる。

2.5.2. クライアント上でプレイヤーの動きをテストする

クライアント上でプレイヤーの動きをテストするには、ゲームのインスタンスが 2 つ同時に実行されている必要があり、そのうちのひとつがホストになる。そのため、現在のプロジェクトをビルド後実行する必要がある。

ひとつの PC 上で、エディタ上で動作するプレイヤーと、スタンドアロンのプレイヤーが開始され、この間で通信をテストすることができる。どちらかのプレイヤーで NetworkManagerHUD で表示されるゲーム内 UI から Host ボタンをクリックすると、このゲームがホストとして開始され、NetworkManager によってプレイヤーゲームオブジェクトが作成される。WASD キーまたは方向キーでプレイヤーの移動と方向転換を行えることを確認する。

シーン内でプレイヤーゲーオブジェクトを動かした後に、Unity に戻り、ゲームをエディター上で実行する。

ゲームがエディター上で実行され、同様に NetworkManagerHUD によりゲーム内 UI が表示され、LAN Client ボタンをクリックすると、クライアントとしてホストに接続される。

ここで、同じ入力によって、2 つのプレイヤーゲーオブジェクトの両方がシーン内で動くことを確認できる。その後、スタンドアロンプレイヤーに切り替えると、プレイヤーゲーオブジェクトの位置が、ホスト上とクライアント上で異なる。

これは、PlayerController スクリプトがネットワークを認識しないためで、この段階では、2 つのプレイヤーゲーオブジェクトの両方に同じスクリプトが添付されており、2 つは同じ入力を、ゲームの別々のインスタンス内で処理している。

ホストとクライアントは互いに認識し合っており、NetworkManager によってプレイヤーゲーオブジェクトが各インスタンス内に 1 つずつ、それぞれのプレイヤー用に作成されているが、どちらのプレイヤーゲーオブジェクトもホストとの通信を行っていないため、それらの位置は NetworkManager によってトラッキングされておらず、したがって同期されていない