オンラインゲーム向けミドルウェアを

用いたPCとスマートフォンの画面共有

N12-009　伊野 晃生

目次

1　背景と目的 ･･･････････ -3-

2　方法　･･･････････ -3-

　2.1　開発環境・使用ソフトウェア　･･･････････ -3-

　2.2　画面共有技術習得のためのサンプル概要　･･･････････ -3-

　2.3　Unity 2D Shootingチュートリアルのオリジナルサンプル概要　･･････････ -8-

3　結果・考察　･･･････････ -14-

　3.1　結果　･･･････････ -14-

　3.2　改善点・考察　･･･････････ -14-

4　参考文献　･･･････････ -15-

5　謝辞　･･･････････ -15-

　付録　･･･････････ -16-

1　背景と目的

　VR技術を用いているアプリを使う際、HMD(Head Mounted Display)を使用するが、HMDの画面は基本的に外部からは見る事ができない。さらに近隣の小学校を訪問し、iPadやiPodtouch向けアプリを子供達に遊んでもらうイベントを行った。保護者の方から、子供が今どのような画面を見ているのかわからない、子供と同じ画面を見て楽しさを共有したい、といった意見を受けた。それに加え私達も、画面が見えないため一度HMDを外してもらい覗き込む、といった行動が必要となり、子供がストレスを感じてしまうということがあった。PC向けHMDではHMDがPCにつながれているため、PCのモニターなどに画面を映し出し画面共有を行うことはできる[1][2]。しかし、スマートホン向けHMDでは有線でスマートホン同士の画面共有を行うことは難しい。

　これらのことから、タブレット端末に映る画面を使用する距離に関係なく別のタブレット端末で見ることができる画面共有技術を、オンラインゲームではプレイヤーの位置情報や同じ画面を表示させていることから、オンラインゲームに使用されているオンラインゲーム向けミドルウェアを用いて実現することを本研究の目的とした。

2　方法

2.1　開発環境・使用ソフトウェア

　アプリの開発対象端末として、どのような環境でも利用することができ、通信することが可能なスマートホン、タブレット端末、PCを選択した。開発環境は、Windows8のOS、アプリ開発のUnity[3]、コーディングの開発言語はC#、GMOクラウド株式会社がサービスしているオンラインゲーム向けミドルウェアのUnity版AssetであるPUN(Photon Unity Network)と、サーバ構築不要のPhoton Cloudサービス[4]を用いた。

　画面共有技術のサンプル[5]、Unity 2D Shootingチュートリアルのオリジナルサンプルを製作するにあたり、ミドルウェア部分ではPhotonサービスを使用するユーザ識別IDの設定を行い、利用する地域と接続するIPアドレスの設定などを行った。

　アプリ部分では、PUNに含まれている画面共有用のPhotonViewと呼ばれるコンポーネントを、視界を映すオブジェクトであるカメラに付け加え、操作するプレイヤーやカメラなどの動く必要のあるオブジェクトに関するコード部分を、PUNに含まれているPhotonNetworkクラスを使用したものに全て書き換え、Photon Cloudを利用できる状態までアプリに実装を行った。またオリジナルサンプルには、サンプルとして簡単である2Dであること、iOSでも使用可能なこと、PUNを利用できること、を満たすものとして、Unity社が無料公開しているUnity 2D Shootingチュートリアル[6]をベースとして使用した。

2.2　画面共有技術習得のためのサンプル概要

　こちらはiOSに実装しないためタッチ操作の機能を付けずに製作した。実行すると、図1のような画面が映る。アプリを実行し、1台目が起動した物にはキャラクターを動かすことのできるプレイヤー機として割り当てた。2台目以降起動した物には観客機を割り当て、操作は一切できないようにした。

　図1の左画面はプレイヤー機の物であるが、プレイヤー機であることがわかるように表示されている。さらに、キーボードの右矢印キーを押すと右に、キーボードの左矢印キーを押すと左にキャラクターが動くようにした。そして、キーボードのAキーを押すとキャラクターではなく、この画面を映しているカメラのオブジェクトが左方向に動き、キーボードのDキーを押すとカメラのオブジェクトが右方向に動く。

　一方、観客機はプレイヤー機のような操作はできず、観客機の画面はプレイヤー機のカメラと同じであるため、プレイヤー機でカメラを動かすと観客機の画面も動くという仕様である。

　Photon CloudのInspector画面を図2に示す。HostingでPhoton Cloudサービスを利用するという選択をした。Regionでは利用地域の設定を行い、使用する地域は日本のためJpを選択した。AppIdでは利用するユーザの識別IDを入力する。図2にあるIDは私の取得したIDである。複雑な動きをするアプリではないため、Protocolは安定性よりも処理速度に優れているUDP(User Datagram Protocol)を選択した。

　Client SettingのAuto-Join Lobbyは本来であればサーバに接続して、マスターサーバと呼ばれる部屋を振り分けるためのサーバに接続するためには手動で設定しなければならないが、チェックを入れることでこの作業を短縮することができるのでチェックを入れて利用することにした。Enable Lobby Statsは現在のマスターサーバの状況をクライアント側から見る事ができる機能だが、今回はその機能を利用する必要がないためチェックを外している。Rpc Listも主にサーバ側の設定であるため、この3つある項目は何も設定していない。

　図3はキャラクターのInspectorの画面である。Transformで初期位置の設定を行う。地面をPositionのY座標1に作っているため、少し高い位置に生成されるようにY座標を2にしている。Sprite Rendererではキャラクターの画像を設定している。Circle Collider 2DとRigidbody 2Dで物理演算の設定を行い、地面をすり抜けて落ちないように設定している。

　Player Controllerコンポーネントでは、Player Controllerスクリプトを設定しており、その中に記述されている変数SPEEDでキャラクターを操作したときの動く速度を設定している。

　Photon Viewコンポーネントでは、設定する部分があまりなく、唯一設定する必要があるのは、どのオブジェクトを画面共有するかというObserved Componentsの部分である。この部分にプレイヤーのオブジェクトであるPlayerPrefabに設定されているSynchronizerスクリプトをPhoton Viewコンポーネントに付けている。

　Synchronizerコンポーネントには、オブジェクトとそのオブジェクトが存在している位置情報を別のSynchronizerコンポーネントと共有するという機能がある。

　またNetworkManagerオブジェクトがあり、ここで通信やサーバ接続の設定を記述したNetworkManager.vsをアタッチしている。

　図4は画面を映すカメラのInspector画面である。Transformの意味は図3の物と同じであるが、画面を映すためプレイヤーと近すぎると映すことはできない。この理由から少し引いて映すように、奥行きを表すZ座標を-10に設定して図1のような視界を映すようにしている。

　Cameraコンポーネントは何も設定する必要がないため、初期設定のままである。GUI Layer、Flare Layer、Audio Listenerも同様に初期設定である。

　Rigidbody 2DとCircle Collider 2Dコンポーネントはプレイヤーオブジェクトと同じで、重力落下させない、かつ移動を可能にするために採用している。

　Camera Controllerコンポーネントでも変数SPEEDでカメラを動かす速度を設定できるようにしている。Photon Viewコンポーネントもプレイヤーオブジェクトと同様に、画面共有をカメラも行う必要があるため設定している。なおObserved Componentsはカメラが画面共有を行うオブジェクトの対象が必要ではなく、カメラ自体を共有するだけでいいので設定していない。

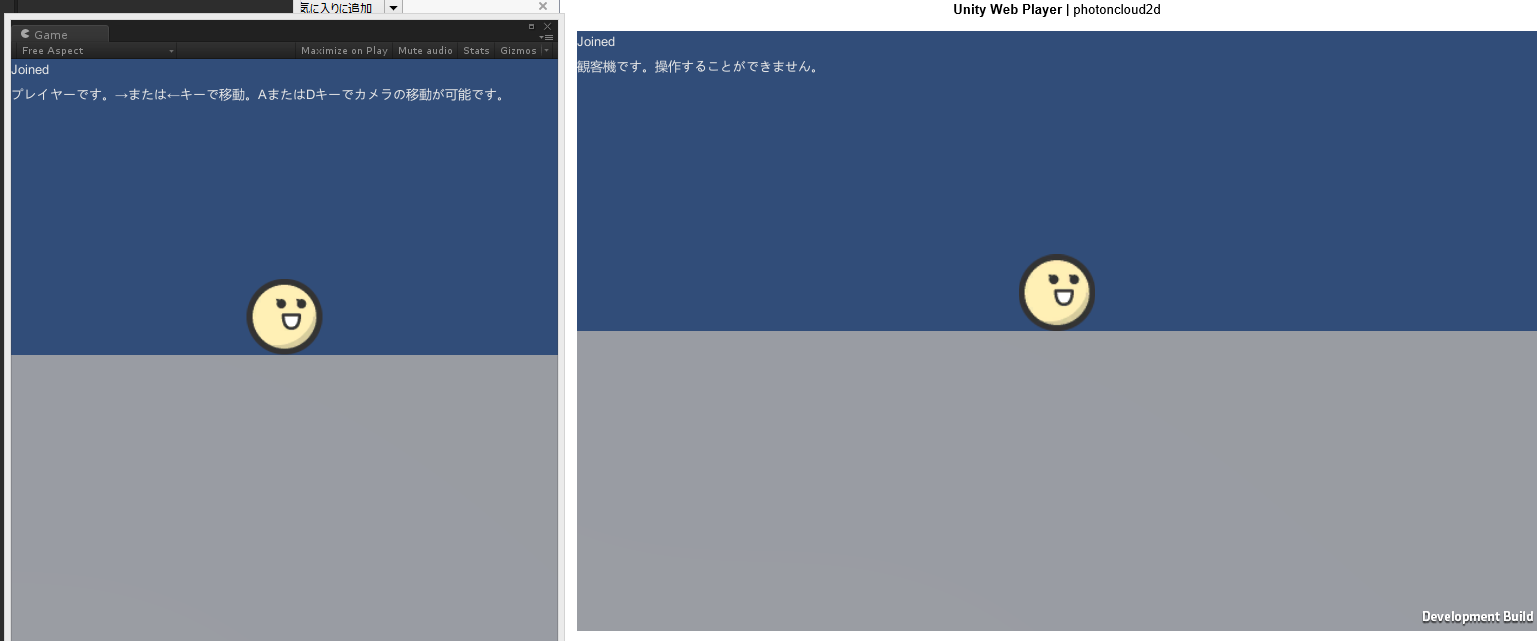


図1　実際の画面　左はプレイヤー機の画面、右は観客機の画面

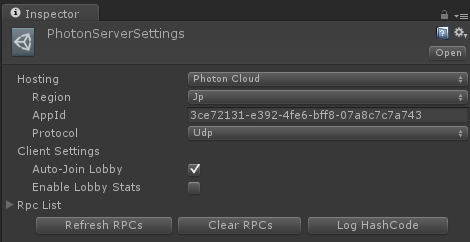


図2　Photon Cloud設定画面

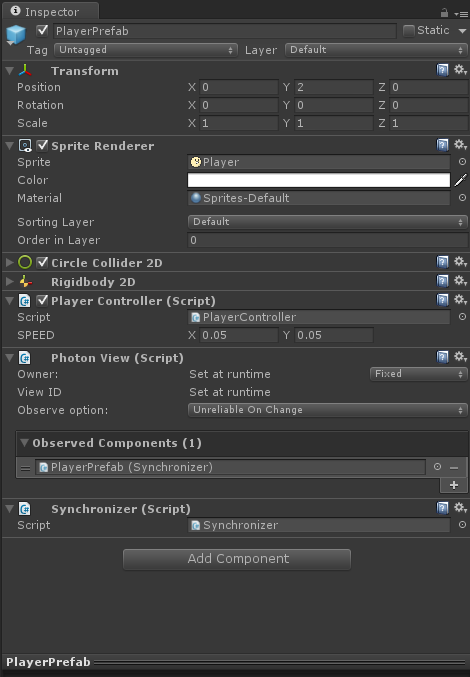


図3　プレイヤーオブジェクト設定画面

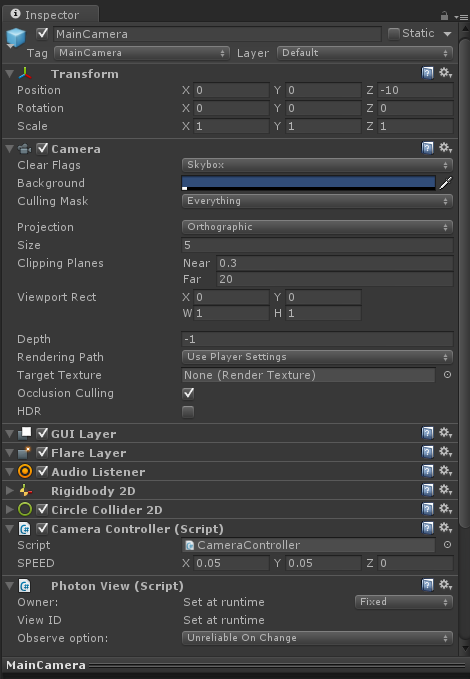


図4　カメラオブジェクト設定画面

2.3　Unity 2D Shootingチュートリアルのオリジナルサンプル概要

　こちらは実際の画面共有の使用感を味わう必要があるため、iOSデバイスで動作できるためにタッチ操作を加えた。このオリジナルサンプルも同様に、起動した1台目は操作可能なプレイヤー機に割り当てる。2台目以降に起動したクライアントは観客機に割り当て、操作ができない仕様にしている。

　図5は、プレイヤー機側の画面である。元々あるUnity 2D Shootingチュートリアルから、右に移動する→ボタン、ゲームスタートするスタートボタンを実装した。スタートボタンにはStartButton.csがアタッチされている。カメラ視野は、本来であれば移動可能範囲全てを映すように設定されているが、あえてカメラ範囲を狭くするように設定した。さらにプレイヤー機か観客機かを見えるようにした。また、1000万分の1秒を表すticksの数値をプレイヤー機で取得してから、観客機に取得したタイミングの信号を送信し、信号を受け取った瞬間に観客機のticksを取得するようにした。そしてプレイヤー機と観客機で取得したticksの差分を取ることで遅延時間を計算できる仕様に作り替えた。

　図6の上から、プレイヤー機での取得ticksの値、観客機での取得ticksの値、二つの取得したticksの差分、差分のticks数は4ケタ以上で表示され、それを秒単位に換算するために1万で割り、遅延時間を計算した物、の4つを表示させるようにした。右上にはそのゲーム中のスコア、右下にはその端末での最高記録のハイスコアが表示される。また、スタートボタンを押すと、Shooting Game Press Xの表示は消えてゲームが始まりプレイヤーが生成される。

　図7は、観客機の画面だが図5とあまり変わらず、図6と類似しており図8では観客機のため、操作はできないがスタートボタンは押せる仕様である。しかしプレイヤー機で生成されたプレイヤーキャラクターは観客機にも映り、プレイヤー機で右に動かすと、観客機の画面でも右に動き、画面共有は行えるようになっている。

　またNetworkManagerオブジェクトがあり、ここで通信やサーバ接続の設定を記述したManager.csをアタッチしている。

　図9は、プレイヤーオブジェクトのInspector画面である。Transformは位置情報があるコンポーネントである。敵キャラクターは最初、画面上端から出現し、下方向に向かって移動するので、画面中央ではなく、少し下部分からゲームスタートするように上下の位置を示すY座標を-1.5に設定した。なお、下方向の移動可能範囲は-3までである。

　Sprite Rendererはプレイヤーキャラクターの物を使用し、Animatorコンポーネントでアニメーションとして動くようにしている。Rigidbody 2Dコンポーネントは物理演算を利用可能にするために付け加えた。

　PlayerコンポーネントではPlayer.csをアタッチしており、プレイヤーキャラクターの移動に関するコードを記述している。Spaceshipコンポーネントは敵キャラクターにもアタッチしていて、プレイヤーや敵キャラクターの放つ弾に当たった時の判定や動作を記述している。

　Photon Viewコンポーネントは、2.2であるものと同じで画面共有を行う対象を設定するコンポーネントである。ここでは、Observed Componentsでプレイヤーオブジェクトを共有するように設定している。

　Synchronizerコンポーネントも2.2同様に、画面共有したいオブジェクトの位置情報を通信するためのコードが記述されている。Right Moveでは、→ボタンを押すと右方向にプレイヤーキャラクターが動くようなコードが記述されているコンポーネントである。

　図10は、カメラオブジェクトのInspector画面である。Transformでは、プレイヤーキャラクターをカメラで中心に捉えて画面に映す必要があるため、プレイヤーと同じY座標-1.5に初期位置を設定した。カメラは少し引いた位置から画面を映す必要があるため、奥行きを設定しているZ座標を-4にした。Camera、Flare Layer、GUI Layerコンポーネントは何も設定する必要がないため、初期設定のままである。Audio LIstenerは全ての音を拾う必要があるため必須のコンポーネントである。

　Cameracontrolコンポーネントは、Cameracontrol.csがアタッチされていて、カメラの挙動をコードで記述している。またこのコードではプレイヤーが右に動いた分、同じだけカメラも動くようにしており、常にプレイヤーキャラクターを中心に映し続けることができるようになっている。

　Photon Viewコンポーネントは上記の通りで画面共有のコンポーネントである。Observed Componentsにはカメラオブジェクトが設定してある。今回はカメラを初期から配置しているのではなく、ゲーム起動が確認できてからカメラを生成するような仕様にしているのでこのようにしている。

　Synchronizerコンポーネントは上記と同様に、オブジェクトの位置情報を通信するためのコードが記述されているSynchronizer.csがアタッチされている。

　カメラを移動させるためにRigidbody 2Dコンポーネントも付け加えている。



図5　プレイヤー機画面



図6 プレイヤー機画面詳細



図7　観客機画面

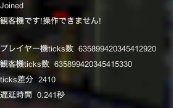


図8 観客機画面詳細

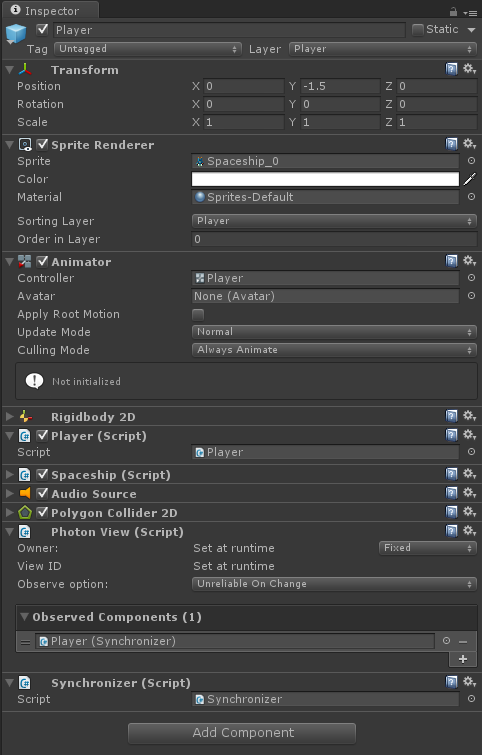


図9　プレイヤーオブジェクト設定画面

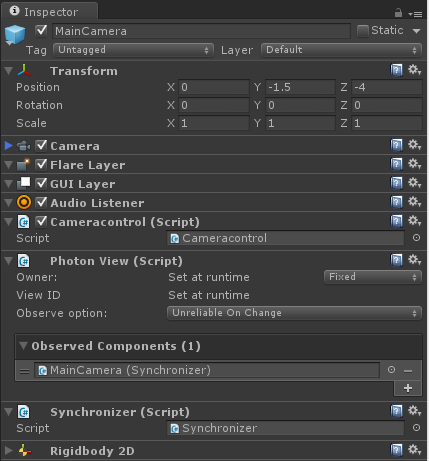


図10　カメラオブジェクト設定画面

3　結果・考察

3.1　結果

　成果物としては、画面共有技術の習得のためのサンプル、Unity 2D Shootingチュートリアルをベースとしたオリジナルアプリの2点を製作した。

　2.2のサンプルではPhoton Cloudの使用、運用方法を学んだが、C#のコーディングとはまた別にPUN独自のクラスやコンポーネントを知り、どのような動作をするものかを理解した。画面共有をこのサンプルで体感したが、キャラクターがただ動くだけだったので使用感などはあまり気にはならなかった。

　2.3のオリジナルサンプルを製作したが、まず気になったのは2.2のサンプルでは気にならなかった遅延であった。2.3サンプルは元々のベースはゲームアプリなので、リアルタイムで動いていくこのサンプルでは少しの遅延でも大きく感じた。その遅延を感覚で感じるだけでなく、数字として可視化できるようにしたが、製作で使用したPCで接続可能な学内LAN2種類、研究室内LAN2種類の計4種類の回線を使用して画面共有を行い、遅延時間の最大は約0.26秒程であった。体感としては少し遅延があり気にはならない程度だが、画面共有した画面ではカクつきが見られ、見る側は少し不快感があると感じた。

3.2　改善点・考察

　2.3のサンプルにおいても最低限の画面共有部分はできたが、当初の遅延時間は約0.28秒程であり、ソースコードのリファクタリングで約0.02秒の遅延時間を短縮することができた。また、Photon Cloudの技術面においても遅延時間を短くできる部分があると思い、クライアント間での通信回数を増加させることでカクつきを減らし、目標としていたVRアプリへの実装ができれば、不快感を軽減してより良いアプリに仕上がるのではないかと感じた。

　今回、サンプルを二つ製作したが、やはり無線を使用した画面共有には遅延が出る。今回の結果では最大約0.26秒の遅延があり、敵機も画面共有を組み込み、よりリアルタイムの通信を感じることができるようになることで、実際に保護者の方に使用してもらい使用感はどうなのか、などイベントを通して意見を聞き、さらに良い使用感に仕上げる必要があるだろう。

4　参考文献

[1]　梶 克彦、平田圭二(2008) 「t-Roomにおけるデスクトップ画面共有機能の実装とその応用」

<http://www.echo.nuee.nagoya-u.ac.jp/~kaji/pdf/jssst2008-kaji-desktop.pdf>

[2]　野口 卓馬、 井口 信和(2014) 「ミーティング支援システムにおけるHTML5を利用した描画画面共有機能の開発」

<http://www.jsise.org/taikai/2014/program/contents/pdf/G1-1.pdf>

[3]　Unity Scripting API

<http://docs.unity3d.com/ScriptReference/>

[4]　Photon Realtimeサービス (PhotonCloud)

<https://www.photonengine.com/ja/Realtime>

[5]　Unity × PhotonCloud 2Dオンラインアプリチュートリアル

<http://dev.classmethod.jp/etc/unity-photon-2d-physics-online-game-01/>

[6]　Unity 2D Shooting チュートリアル - Github

<https://github.com/unity3d-jp-tutorials/2d-shooting-game/wiki>

5　謝辞

　本研究に際して、様々なご指導、ご指摘を頂きました矢野浩二郎先生に深謝致します。また、研究試行の際にご指摘を頂きました矢野研究室の同期の皆様に感謝致します。

付録

　2.2　NetworkManager.cs

using UnityEngine;  
using System.Collections;  
  
public class NetworkManager : Photon.MonoBehaviour {  
  
 public GameObject player;  
 public static int flag;  
 public GameObject str = null;  
   
 void Awake(){  
 PhotonNetwork.ConnectUsingSettings ("v0.1");  
 PlayerController.pflag = 2;  
 }  
   
 void Update () {   
 }  
  
 void OnJoinedLobby(){  
 PhotonNetwork.JoinRandomRoom ();  
 flag = 1;  
 }  
  
 void OnPhotonRandomJoinFailed(){  
 PhotonNetwork.CreateRoom (null);  
 flag = 0;  
 }  
  
 void OnJoinedRoom(){  
 Vector3 spawnPosition = new Vector3 (0, 2, 0);  
 Vector3 CspawnPosition = new Vector3 (0, 0.0001f, -10);  
  
 if (flag == 0) {  
 var obj = PhotonNetwork.Instantiate ("PlayerPrefab", spawnPosition, Quaternion.identity, 0);  
 obj.name = "PlayerPrefab1";  
 flag = 1;  
  
 var cam = PhotonNetwork.Instantiate ("MainCamera", CspawnPosition, Quaternion.identity, 0);  
 cam.name = "MainCamera1";  
  
 PlayerController.pflag = 1;  
 CameraController.cflag = 1;  
 } else {  
 PlayerController.pflag = 0;  
 CameraController.cflag = 0;  
 }  
 }  
  
 void OnGUI(){  
 GUILayout.Label (PhotonNetwork.connectionStateDetailed.ToString ());  
  
 if (PlayerController.pflag == 1) {  
 GUILayout.Label ("プレイヤーです。→または←キーで移動。AまたはDキーでカメラの移動が可能です。");  
 } else if (PlayerController.pflag == 0) {  
 GUILayout.Label ("観客機です。操作することができません。");  
 } else if (PlayerController.pflag == 2) {  
 GUILayout.Label ("待機中...");  
 }  
 }  
}

　2.2　PlayerController.cs

using UnityEngine;  
using System.Collections;  
  
public class PlayerController : MonoBehaviour {  
 // 速度  
 public Vector2 SPEED = new Vector2(0.05f, 0.05f);  
 public static int pflag = 0;  
 // Use this for initialization  
 void Start () {  
 }  
   
 // Update is called once per frame  
 void Update () {  
 if (pflag == 1) {  
 PlayerMove ();  
 }  
 }  
 // 移動関数  
 void PlayerMove(){  
 // 現在位置をPositionに代入  
 Vector2 Position = transform.position;  
 // 左キーを押し続けていたら  
 if(Input.GetKey("left")){  
 // 代入したPositionに対して加算減算を行う  
 Position.x -= SPEED.x;  
 } else if(Input.GetKey("right")){ // 右キーを押し続けていたら  
 // 代入したPositionに対して加算減算を行う  
 Position.x += SPEED.x;  
 }  
 // 現在の位置に加算減算を行ったPositionを代入する  
 transform.position = Position;  
 }  
}

　2.2　CameraController.cs

using UnityEngine;  
using System.Collections;  
  
public class CameraController : MonoBehaviour {  
  
 public Vector3 SPEED = new Vector3(0.05f, 0.05f, 0);  
 public static int cflag = 0;  
  
 void Start () {  
 }  
  
 void Update () {  
 if (cflag == 1) {  
 CameraMove ();  
 }  
 }  
  
 void CameraMove(){  
 // 現在位置をPositionに代入  
 Vector3 Position = transform.position;  
 // 左キーを押し続けていたら  
 if (Input.GetKey (KeyCode.A)) {  
 // 代入したPositionに対して加算減算を行う  
 Position.x -= SPEED.x;  
 } else if (Input.GetKey (KeyCode.D)) { // 右キーを押し続けていたら  
 // 代入したPositionに対して加算減算を行う  
 Position.x += SPEED.x;  
 }  
 // 現在の位置に加算減算を行ったPositionを代入する  
 transform.position = Position;  
 }  
   
}

　2.2　Synchronizer.cs

using UnityEngine;  
using System.Collections;  
  
public class Synchronizer : Photon.MonoBehaviour {  
  
 void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info){  
 if (stream.isWriting) {  
 stream.SendNext (GetComponent<Transform>().position);  
 stream.SendNext (GetComponent<Transform>().rotation);  
 stream.SendNext (GetComponent<Rigidbody2D>().velocity);  
 } else {  
 GetComponent<Transform>().position = (Vector3)stream.ReceiveNext ();  
 transform.rotation = (Quaternion)stream.ReceiveNext ();  
 GetComponent<Rigidbody2D>().velocity = (Vector2)stream.ReceiveNext ();  
 }  
 }  
}

　2.3　Manager.cs

using UnityEngine;  
using System.Collections;  
  
public class Manager : MonoBehaviour  
{  
  
 public static int mainflag;  
 public GameObject player;  
 public GameObject MainCamera;  
 public static Vector3 CspawnPosition = new Vector3 (0.0f, -1.5f, -4.0f);  
 public static Vector3 PspawnPosition = new Vector3 (0.0f, -1.5f, 0.0f);  
 public static bool flag = false;  
  
 // タイトル  
 public GameObject title;  
  
 void Awake(){  
 PhotonNetwork.ConnectUsingSettings ("v0.1");  
  
 }  
  
 void OnJoinedLobby(){  
 PhotonNetwork.JoinRandomRoom ();  
 mainflag = 1;  
 }  
  
 void OnPhotonRandomJoinFailed(){  
 PhotonNetwork.CreateRoom (null);  
 mainflag = 0;  
 }  
  
 void OnJoinedRoom(){  
 if (mainflag == 0) {  
 title = GameObject.Find ("Title");  
  
 Player.pflag = 1;  
 Cameracontrol.cflag = 1;  
   
 var cam = PhotonNetwork.Instantiate ("MainCamera", CspawnPosition, MainCamera.transform.rotation, 0);  
 cam.name = "Camera";  
 flag = false;  
 } else {  
 }  
 }  
   
 void Update (){  
 if (IsPlaying () == false && flag == true) {  
 Application.LoadLevel ("Stage");  
 }  
  
 if (IsPlaying () == false && StartButton.SB == 1 ) {  
 GameStart ();  
 flag = true;  
 StartButton.SB = 0;  
 }  
  
 }

void GameStart ()  
 {  
 // ゲームスタート時に、タイトルを非表示にしてプレイヤーを作成する  
 title.SetActive (false);  
  
 if (mainflag == 0) {  
 var ply = PhotonNetwork.Instantiate ("Player", PspawnPosition, Quaternion.identity, 0);  
 ply.name = "PlayeR";  
 }  
  
 }  
   
  
 public void GameOver ()  
 {  
 FindObjectOfType<Score>().Save();  
 // ゲームオーバー時に、タイトルを表示する  
 title.SetActive (true);  
 }  
   
 public bool IsPlaying ()  
 {  
 // ゲーム中かどうかはタイトルの表示/非表示で判断する  
 return title.activeSelf == false;  
 }  
  
 void OnGUI(){  
 GUILayout.Label (PhotonNetwork.connectionStateDetailed.ToString ());  
 if (mainflag == 0) {  
 GUILayout.Label ("プレイヤー機です!操作できます!");  
 } else if (mainflag == 1) {  
 GUILayout.Label ("観客機です!操作できません!");  
 }  
 GUILayout.Space (20);  
 GUILayout.Label ("プレイヤー機ticks数 " + Synchronizer.oldTicks);  
 GUILayout.Label ("観客機ticks数 " + Synchronizer.newTicks);  
 GUILayout.Label ("ticks差分 " + Synchronizer.ddiff);  
 GUILayout.Label ("遅延時間 " + Synchronizer.Diff + "秒");  
 }  
}

　2.3　Player.cs

using UnityEngine;  
using System.Collections;  
  
public class Manager : MonoBehaviour  
{  
  
 public static int mainflag;  
 public GameObject player;  
 public GameObject MainCamera;  
 public static Vector3 CspawnPosition = new Vector3 (0.0f, -1.5f, -4.0f);  
 public static Vector3 PspawnPosition = new Vector3 (0.0f, -1.5f, 0.0f);  
 public static bool flag = false;  
  
 // タイトル  
 public GameObject title;  
  
 void Awake(){  
 PhotonNetwork.ConnectUsingSettings ("v0.1");  
  
 }  
  
 void OnJoinedLobby(){  
 PhotonNetwork.JoinRandomRoom ();  
 mainflag = 1;  
 }  
  
 void OnPhotonRandomJoinFailed(){  
 PhotonNetwork.CreateRoom (null);  
 mainflag = 0;  
 }  
  
 void OnJoinedRoom(){  
 if (mainflag == 0) {  
 title = GameObject.Find ("Title");  
  
 Player.pflag = 1;  
 Cameracontrol.cflag = 1;

var cam = PhotonNetwork.Instantiate ("MainCamera", CspawnPosition, MainCamera.transform.rotation, 0);  
 cam.name = "Camera";  
 flag = false;  
 } else {  
 }  
 }  
   
 void Update (){  
 if (IsPlaying () == false && flag == true) {  
 Application.LoadLevel ("Stage");  
 }  
  
 if (IsPlaying () == false && StartButton.SB == 1 ) {  
 GameStart ();  
 flag = true;  
 StartButton.SB = 0;  
 }  
  
 }  
   
 void GameStart ()  
 {  
 // ゲームスタート時に、タイトルを非表示にしてプレイヤーを作成する  
 title.SetActive (false);  
  
 if (mainflag == 0) {  
 var ply = PhotonNetwork.Instantiate ("Player", PspawnPosition, Quaternion.identity, 0);  
 ply.name = "PlayeR";  
 }  
  
 }  
   
  
 public void GameOver ()  
 {  
 FindObjectOfType<Score>().Save();  
 // ゲームオーバー時に、タイトルを表示する  
 title.SetActive (true);  
 }

public bool IsPlaying ()  
 {  
 // ゲーム中かどうかはタイトルの表示/非表示で判断する  
 return title.activeSelf == false;  
 }  
  
 void OnGUI(){  
 GUILayout.Label (PhotonNetwork.connectionStateDetailed.ToString ());  
 if (mainflag == 0) {  
 GUILayout.Label ("プレイヤー機です!操作できます!");  
 } else if (mainflag == 1) {  
 GUILayout.Label ("観客機です!操作できません!");  
 }  
 GUILayout.Space (20);  
 GUILayout.Label ("プレイヤー機ticks数 " + Synchronizer.oldTicks);  
 GUILayout.Label ("観客機ticks数 " + Synchronizer.newTicks);  
 GUILayout.Label ("ticks差分 " + Synchronizer.ddiff);  
 GUILayout.Label ("遅延時間 " + Synchronizer.Sdiff + "秒");  
 }  
}

　2.3　CameraControl.cs

using UnityEngine;  
using System.Collections;  
  
public class Cameracontrol : MonoBehaviour {  
   
 int xMoveLimitMax = 4;  
 int xMoveLimitMin = -4;  
 int yMoveLimitMax = 3;  
 int yMoveLimitMin = -3;  
 public static int cflag;  
 public static bool cpush = false;  
  
 void Start () {  
 }  
  
 public void PushDown(){  
 cpush = true;  
 }

public void PushUp(){  
 cpush = false;  
 }  
  
 void Update () {  
 if (Manager.flag == true) {  
 if (cflag == 1) {  
 if (cpush == true) {  
 CameraMove ();  
 }  
 }  
 }  
 CameraMoveArea ();  
 }  
   
 void CameraMove (){  
   
 // カメラの座標を取得  
 Vector3 cpos = transform.position;  
   
 cpos.x += 0.1f;  
 // 現在の位置に加算減算を行ったPositionを代入する  
 transform.position = cpos;  
   
 }  
  
 void CameraMoveArea(){  
 Vector3 cpos = transform.position;  
 cpos.x = Mathf.Clamp (cpos.x, xMoveLimitMin, xMoveLimitMax);  
 cpos.y = Mathf.Clamp (cpos.y, yMoveLimitMin, yMoveLimitMax);  
 transform.position = cpos;  
 }  
  
}

　2.3　Synchronizer.cs

using UnityEngine;  
using System.Collections;  
  
public class Synchronizer : Photon.MonoBehaviour {  
 public static string ddiff;  
 public static string Diff;

public static long oldTicks;  
 public static long newTicks;  
 public static long diff;  
 public static double diffrence;  
  
 void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info){  
  
 oldTicks = System.DateTime.Now.Ticks;  
 PlayerPrefs.SetString ("datetime", oldTicks.ToString ());  
 string dateString = PlayerPrefs.GetString ("datetime");  
 oldTicks = System.Convert.ToInt64 (dateString);  
   
 newTicks = System.DateTime.Now.Ticks;   
 diff = newTicks - oldTicks;  
 ddiff = diff.ToString();  
 diffrence = (double) diff / 10000;  
 Diff = diffrence.ToString();  
  
 if (stream.isWriting) {  
 stream.SendNext (GetComponent<Transform>().position);  
 stream.SendNext (GetComponent<Transform>().rotation);  
 stream.SendNext (GetComponent<Rigidbody2D>().velocity);  
 } else {  
 GetComponent<Transform>().position = (Vector3)stream.ReceiveNext ();  
 transform.rotation = (Quaternion)stream.ReceiveNext ();  
 GetComponent<Rigidbody2D>().velocity = (Vector2)stream.ReceiveNext ();  
 }  
 }  
}

　2.3　StartButton.cs

using UnityEngine;  
using System.Collections;  
  
public class StartButton : MonoBehaviour {  
  
 public static int SB;  
  
 // Use this for initialization  
 void Start () {  
 SB = 0;  
 }  
  
 public void ButtonPush(){  
 SB = 1;  
 }  
}