EXIT GAMES

PhotonNetwork Plugin for Unity3d

v1.11

Contents

Exit Games GmbH, Hongkongstr. 7, 20457 Hamburg, Germany
EU Headquarters: +49 40 413596-0

Exit Games Inc., One World Trade Center, 121 SW Salmon St., Suite 1100, Portland, OR. 97204

www.exitgames.com

1 Introduction

PhotonNetwork is a networking plugin for Unity, which makes Photon easier to use than ever before. Full source code is available, so you can scale this plugin to support any type of multiplayer game you'd ever need.

The PhotonNetwork API looks very similar to that of Unity's networking solution and users who have experience with Unity Networking should feel at home immediately. Even better: An automatic converter will help you port your Unity networking project to the Photon equivalent. Users that have no previous networking experience should have no easier experience than starting with Photon: the powerful API abstracts all the complicated work.

By default, this plugin makes use of the hosted "Exit Games Cloud" service, which runs Photon for you. A setup window registers you (for free) in less than a minute.

Most notable features:

- Dead-easy API
- Server available as hosted service (currently free of charge!)
- Partially automatic conversion from Unity Networking to PhotonNetworking
- Offline mode: re-use your multiplayer code in singleplayer game modes
- Outstanding performance of the Photon Server
- Load balanced workflow scales across servers (with no extra effort)
- No direct P2P and no NAT punch-through needed

2 The Photon Server

2.1 Exit Games Cloud

The Exit Games Cloud is a service that provides hosted and load balanced Photon instances for you, run by Exit Games. Free trials are available and subscription costs for commercial use are comparable with web hosting offers.

In the service, you can't implement your own server logic. Instead, make the clients authoritative.
Applications are separated by "application id" and your client's a "game version". With that, your players won't clash with that of another developer or older versions.

When you imported the editor scripts from the Photon Unity Networking package, a setup window automatically open. Enter your email address and register for the cloud.

2.2 Photon Server SDK

As alternative to the hosted Photon service, you can run your own server and develop on top of our "Load Balancing" game logic. This gives you full control of the server logic.

The Photon v3 SDK can be downloaded on:
http://www.exitgames.com/Download/Photon

If you run your own Photon server, use the setup wizard, to switch your settings for it. Open it in the Menu: Window, Photon Unity Networking.

3 First steps

This plugin consists of quite a few files, however there's only one that truly matters: PhotonNetwork. This class contains all functions and variables that you need. If you ever have custom requirements, you can always modify the source files - this plugin is just an implementation of Photon after all.

The imported package includes a setup wizard, which creates a configuration for either the cloud service or your own Photon server. Check: PhotonServerSettings.

Using Unity Javascript? To be able to use the Photon classes you'll need to move the Plugins folder to the root of your project.

To show you how this API works, here are a few examples right away.

3.1 Connecting to games

PhotonNetwork always uses a master server and one or more game servers. The master server manages the list of game servers and currently running games on those servers. To pick a game (or get into a random one), players connect to the Master server. The Master forwards the clients to the game servers, where the actual gameplay is done. The servers are all run on dedicated machines - there is no such thing as player-hosted 'servers'. You don't have to bother remembering about the server organization though, as the API all hides this for you.

```
PhotonNetwork. ConnectUsingSettings("v1.0");
```

The code above is required to make use of any PhotonNetwork features. It sets your client's game version and uses the setup-wizard's config (stored in: PhotonServerSettings).
The wizard can also be used when you host Photon yourself. Alternatively, use Connect() and you can ignore the PhotonServerSettings file.

3.1.1 Versioning

The loadbalancing logic for Photon uses your appID to separate your players from anyone else's. The same is done by game version, which separates players with a new client from those with older clients.
As we can't guarantee that different Photon Unity Networking versions are compatible with each other, we add the PUN version to your game's version before sending it (since PUN v1.7).

3.2 Creating and Joining Games

Next, you'll want to join or create a room. The following code showcases some required functions:

```
//Join a room
PhotonNetwork.JoinRoom(roomName);

//Create this room.
PhotonNetwork.CreateRoom(roomName);
// Fails if it already exists and calls: OnPhotonCreateGameFailed

//Tries to join any random game:
PhotonNetwork.JoinRandomRoom();
//Fails if there are no matching games: OnPhotonRandomJoinFailed
```

A list of currently running games is provided by the master server's lobby. It can be joined like other rooms but only provides and updates the list of rooms. The PhotonNetwork plugin will automatically join the lobby after connecting. When you're joining a room, the list will no longer update.

To display the list of rooms (in a lobby):

```
foreach (Room game in PhotonNetwork.GetRoomList())
{
   GUILayout.Label(game.name + " " + game.playerCount + "/" + game.maxPlayers);
}
```

Alternatively, the game can use random matchmaking: It will try to join any room and fail if none has room for another player. In that case: Create a room without name and wait until other players join it randomly.

3.3 MonoBehaviour Callbacks

PhotonNetwork implements several callbacks to let your game know about state changes, like "connected" or "joined a game". Each of the methods used as callback is part of the PhotonNetworkingMessage enum. Per enum item, the use is explained (check the tooltip when you type in e.g. PhotonNetworkingMessage. OnConnectedToPhoton.
You can add these methods on any number of MonoBehaviours, they will be called in the respective situation.
The complete list of callbacks is also in the Plugin reference.

This covers the basics of setting up game rooms. Next up is actual communication in games.

3.4 Sending messages in game rooms

Inside a room you are able to send network messages to other connected players. Furthermore you are able to send buffered messages that will also be sent to players that connect in the future (for spawning your player for instance).

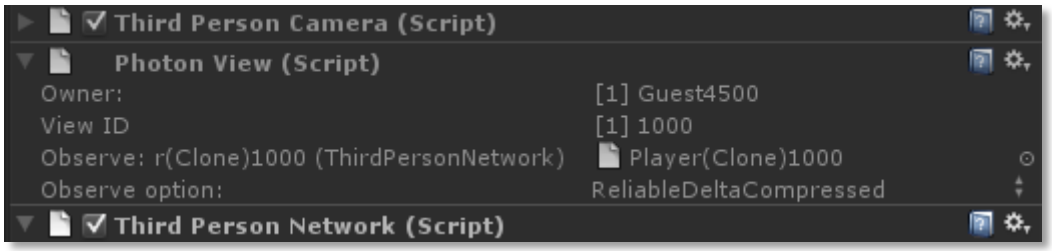
Sending messages can be done using two methods. Either RPCs or by using the PhotonView property OnSerializePhotonView. There is more network interaction though. You can listen for callbacks for certain network events (e.g. OnPhotonInstantiate, OnPhotonPlayerConnected) and you can trigger some of these events (PhotonNetwork.Instantiate). Don't worry if you're confused by the last paragraph, next up we'll explain for each of these subjects.

3.5 PhotonView

PhotonView is a script component that is used to send messages (RPCs and OnSerializePhotonView). You need to attach the PhotonView to your games gameobjects. Note that the PhotonView is very similar to Unity's NetworkView.

At all times, you need at least one PhotonView in your game in order to send messages and optionally instantiate/allocate other PhotonViews.

To add a PhotonView to a gameobject, simply select a gameobject and use: "Components/Miscellaneous/Photon View".



3.5.1 Observe Transform

If you attach a Transform to a PhotonView's Observe property, you can choose to sync Position, Rotation and Scale or a combination of those across the players. This can be a great help for prototyping or smaller games.
Note: A change to any observed value will send out all observed values - not just the single value that's changed. Also, updates are not smoothed or interpolated.

3.5.2 Observe MonoBehaviour

A PhotonView can be set to observe a MonoBehaviour. In this case, the script's OnPhotonSerializeView method will be called. This method is called for writing an object's state and for reading it, depending on whether the script is controlled by the local player.

The simple code below shows how to add character state synchronization with just a few lines of code more:

```
void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
  if (stream.isWriting)
  {
     //We own this player: send the others our data
     stream.SendNext((int)controllerScript._characterState);
     stream.SendNext(transform.position);
     stream.SendNext(transform.rotation);
  }
  else
  {
     //Network player, receive data
     controllerScript._characterState = (CharacterState)(int)stream.ReceiveNext();
     correctPlayerPos = (Vector3)stream.ReceiveNext();
     correctPlayerRot = (Quaternion)stream.ReceiveNext();
  }
}
```

Now on, to yet another way to communicate: RPCs.

3.6 Remote Procedure Calls

Remote Procedure Calls (RPCs) are exactly what the name implies: methods that can be called by any client within a room.
To enable remote calls for a method of a MonoBehaviour, you must apply the attribute: [RPC]. A PhotonView instance is needed on the same GameObject, to call the marked function.

```
[RPC]
void ChatMessage(string a, string b)
{
   Debug.Log("ChatMessage " + a + " " + b);
}
```

To call the method from any script, you need access to a PhotonView object. If your script derives from Photon.MonoBehaviour, it has a photonView field. Any regular MonoBehaviour or GameObject can use: PhotonView.Get(this) to get access to its PhotonView component and then call RPCs on it.

```
PhotonView photonView = PhotonView.Get(this);
photonView.RPC("ChatMessage", PhotonTargets.All, "jup", "and jup!");
```

So, instead of directly calling the target method, you call RPC() on a PhotonView. Provide the name of the method to call, which players should call the method and then provide a list of parameters.

Careful: The parameters list used in RPC() has to match the number of expected parameters!
If the receiving client can't find a matching method, it will log an error.
There is one exception to this rule: The last parameter of a RPC method can be of type PhotonMessageInfo, which will provide some context for each call.

```
[RPC]
void ChatMessage(string a, string b, PhotonMessageInfo info)
{
   Debug.Log(String.Format("Info: {0} {1} {2}", info.sender, info.photonView,
info.timestamp));
}
```

4 Plugin reference

The following list sums up all the exposed functions and variables of PhotonNetwork, which is the core of this Unity plugin.
This list is just a provisional listing. The reference for the API is going to be built from source (which gets you tooltips in MonoDevelop and Visual Studio).

PhotonNetwork methods:
void Connect(string serverAddress, int port, string uniqueGameID)
void Disconnect()
void InitializeSecurity()
void CreateRoom(string title)
void JoinRoom(string title)
void JoinRandomRoom()
void LeaveRoom()
Room[] GetRoomList()
int AllocateViewID()
void UnAllocateViewID(int ID)
GameObject Instantiate(Object prefab, Vector3 position, Quaternion rotation, int group)
GameObject Instantiate(Object prefab, Vector3 position, Quaternion rotation, int group, object[] data)
int GetPing()
void CloseConnection(PhotonPlayer kickPlayer)
void Destroy(int viewID)
void Destroy(PhotonView view)
void DestroyPlayerObjects(PhotonPlayer player)
void RemoveAllInstantiatedObjects()
void RemoveAllInstantiatedObjects(PhotonPlayer player)
void RPC(PhotonView view, string methodName, PhotonTargets target, params object[] parameters)
void RPC(PhotonView view, string methodName, PhotonPlayer targetPlayer, params object[] parameters)
void RemoveRPCs()
void RemoveRPCs(PhotonView view)
void RemoveRPCsInGroup(int group)
void SetReceivingEnabled(int group, bool enabled)
void SetSendingEnabled(int group, bool enabled)
void SetLevelPrefix(int prefix)

PhotonNetwork variables
NetworkLogLevel logLevel = NetworkLogLevel.Full;
string playerName
bool offlineMode
int sendRate
int sendRateOnSerialize
bool autoCleanUpPlayerObjects

PhotonNetwork variables (read only)
bool connected;
ConnectionState connectionState;
PlaytimePeerStates connectionStateDetailed
Room room
PhotonPlayer player
PhotonPlayer masterClient
List<PhotonPlayer> playerList
List<PhotonPlayer> otherPlayers
float time
bool isMasterClient

Monobehaviour callbacks
These are all part of the enum PhotonNetworkingMessage, where they are explained, too.

OnMasterClientSwitched(PhotonPlayer newMasterClient)
OnPhotonCreateGameFailed()
OnPhotonJoinGameFailed()
OnPhotonRandomJoinFailed()
OnJoinedLobby()
OnLeftLobby()
OnCreatedRoom()
OnJoinedRoom()
OnConnectedToPhoton()
OnDisconnectedFromPhoton()
OnFailedToConnectToPhoton()
OnReceivedRoomList()
OnReceivedRoomListUpdate()
OnPhotonPlayerConnected(PhotonPlayer player)
OnPhotonPlayerDisconnected(PhotonPlayer player)
OnPhotonInstantiate(PhotonMessageInfo info)
OnPhotonSerializeView(PhotonStream stream)
OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)

PhotonView Variables
int viewID
bool isSceneView
Object observed
PhotonPlayer owner
ViewSynchronization synchronization;
int group = 0;
int prefix = -1;
object[] instantiationData = null;
Hashtable lastOnSerializeDataSent = null;
OnSerializeTransform onSerializeTransformOption =
OnSerializeTransform.PositionAndRotation;
OnSerializeRigidBody onSerializeRigidBodyOption = OnSerializeRigidBody.All;
bool isMine

PhotonView functions
void RPC(string methodName, PhotonTargets target, params object[] parameters)
void RPC(string methodName, PhotonPlayer targetPlayer, params object[] parameters)

PhotonView Static functions
PhotonView Get(Component component)
PhotonView Get(GameObject gameObj)
PhotonView Find(int viewID)

PhotonPlayer

PhotonMessageInfo
PhotonPlayer sender
PhotonView photonView
float timestamp

PhotonStream
bool isWriting
bool isReading
object ReceiveNext()
void SendNext(object obj)
void Serialize(ref bool myVar)
void Serialize(ref int myVar)
void Serialize(ref string myVar)
void Serialize(ref char myVar)
void Serialize(ref short myVar)
void Serialize(ref float myVar)
void Serialize(ref PhotonPlayer myVar)
void Serialize(ref Vector3 myVar)
void Serialize(ref Vector2 myVar)
void Serialize(ref Quaternion myVar)

Room
String name
int playerCount
int maxPlayers
bool open
bool visible
Hashtable properties

5 Various topics

5.1 Differences to Unity Networking

1. **Host model**
   a. Unity networking is server-client based (NOT P2P!). Servers are run via a Unity client (so via one of the players)
   b. Photon is server-client based as well, but has a dedicated server; No more dropped connections due to hosts leaving.
2. **Connectivity**
   a. Unity networking works with NAT punchthrough to try to improve connectivity: since players host the network servers, the connection often fails due to firewalls/routers etc. Connectivity can never be guaranteed, there is a low success rate.
   b. Photon has a dedicated server, there is no need for NAT punchthrough or other concepts. Connectivity is a guaranteed 100%. If, in the rare case, a

connection fails it must be due to a very strict client side network (a business VPN for example).

3. **Performance**
   a. Photon beats Unity networking performance wise. We do not have the figures to prove this yet but the library has been optimized for years now. Furthermore, since the Unity servers are player hosted latency/ping is usually worse; you rely on the connection of the player acting as server. These connections are never any better then the connection of your dedicated Photon server.
4. **Price**
   a. Like the Unity Networking solution, the Photon Unity Networking plugin is free as well.
   You can subscribe to use Photon Cloud hosting service for your game. Alternatively, you can rent your own servers and run Photon on them. The *free license* enables up to 100 concurrent players. Other licenses cost a one-time fee (as you do the hosting) and lift the concurrent user limits.
5. **Features & maintenance**
   a. Unity does not seem to give much priority to their Networking implementation. There are rarely feature improvements and bugfixes are as seldom. The Photon solution is actively maintained and parts of it are available with source code. Furthermore, Photon already offers more features than Unity, such as the built-in load balancing and offline mode.
6. **Master Server**
   a. The Master Server for Photon is a bit different from the Master Server for plain Unity Networking: In our case, it's a Photon Server that lists room-names of currently played games in so called "lobbies".
   Like Unity's Master, it will forward clients to the Game Server(s), where the actual gameplay is done.

5.2 Instantiating objects over the network

In about every game you need to instantiate one or more player objects for every player. There are various options to do so which are listed below.

5.2.1 PhotonNetwork.Instantiate

PUN can automatically take care of spawning an object by passing a starting position, rotation and a prefab name to the PhotonNetwork.Instantiate method.
Requirement: The prefab should be available directly under a Resources/ folder so that the prefab can be loaded at run time. Watch out with webplayers: Everything in the resources folder will be streamed at the very first scene per default. Under the webplayer settings you can specify the first level that uses assets from the Resources folder by using the "First streamed level". If you set this to your first game scene, your preloader and mainmenu will not be slowed down if they don't use the Resources folder assets.

```
void SpawnMyPlayerEverywhere()
{
   PhotonNetwork.Instantiate("MyPrefabName", new Vector3(0,0,0), Quaternion.identity, 0);
   //The last argument is an optional group number, feel free to ignore it for now.
}
```

5.2.2 Gain more control: Manually instantiate

If don't want to rely on the Resources folders to instantiate objects over the network you'll have to manually Instantiate objects as shown in the example at the end of this section.

The main reason for wanting to instantiate manually is gaining control over *what is downloaded when* for streaming webplayers. The details about streaming and the Resources folder in Unity can be found here.

If you spawn manually, you will have to assign a PhotonViewID yourself, these viewID's are the key to routing network messages to the correct gameobject/scripts. The player who wants to own and spawn a new object should allocate a new viewID using PhotonNetwork.AllocateViewID();. This PhotonViewID should then be send to all other players using a PhotonView that has already been set up (for example an existing scene PhotonView). You will have to keep in mind that this RPC needs to be buffered so that any clients that connect later will also receive the spawn instructions. Then the RPC message that is used to spawn the object will need a reference to your desired prefab and instantiate this using Unity's GameObject.Instantiate. Finally you will need to set setup the PhotonViews attached to this prefab by assigning all PhotonViews a PhotonViewID.

```
void SpawnMyPlayerEverywhere()
{
   //Manually allocate PhotonViewID
   PhotonViewID id1 = PhotonNetwork.AllocateViewID();

   photonView.RPC("SpawnOnNetwork", PhotonTargets.AllBuffered, transform.position,
      transform.rotation, id1, PhotonNetwork.player);
}

public Transform playerPrefab; //set this in the inspector

[RPC]
void SpawnOnNetwork(Vector3 pos, Quaternion rot, PhotonViewID id1, PhotonPlayer np)
{
   Transform newPlayer = Instantiate(playerPrefab, pos, rot) as Transform;

   //Set the PhotonView
   PhotonView[] nViews = go.GetComponentsInChildren<PhotonView>();
   nViews[0].viewID = id1;
}
```

If you want to use asset bundles to load your network objects from, all you have to do is add your own assetbundle loading code and replace the "playerPrefab" from the example with the prefab from your asset bundle.

5.3 Offline mode

Offline mode is a feature to be able to re-use your multiplayer code in singleplayer game modes as well.

Mike Hergaarden: At M2H we had to rebuild our games several times as game portals usually require you to remove multiplayer functionality completely. Furthermore, being able to use the same code for single and multiplayer saves a lot of work on itself.

The most common features that you'll want to be able to use in singleplayer are sending RPCs and using PhotonNetwork.Instantiate. The main goal of offline mode is to disable nullreferences and other errors when using PhotonNetwork functionality while not connected. You would still need to keep track of the fact that you're running a

singleplayer game, to set up the game etc. However, while running the game, all code should be reusable.

You need to manually enable offline mode, as PhotonNetwork needs to be able to distinguish erroneous from intended behaviour. Enabling this feature is very easy:

```
PhotonNetwork.offlineMode = true;
```

You can now reuse certain multiplayer methods without generating *any* connections and errors. Furthermore there is no noticeable overhead. Below follows a list of PhotonNetwork functions and variables and their results during offline mode:

PhotonNetwork.player
The player ID is always -1
PhotonNetwork.playerName
Works as expected.
PhotonNetwork.playerList
Contains only the local player
PhotonNetwork.otherPlayers
Always empty
PhotonNetwork.time
returns Time.time;
PhotonNetwork.isMasterClient
Always true
PhotonNetwork.AllocateViewID()
Works as expected.
PhotonNetwork.Instantiate
Works as expected
PhotonNetwork.Destroy
Works as expected.
PhotonNetwork.RemoveRPCs/RemoveRPCsInGroup/SetReceivingEnabled/SetSendingEnabled/SetLevelPrefix
While these make no sense in Singleplayer, they will not hurt either.
PhotonView.RPC
Works as expected.

Note that using other methods than the ones above can yield unexpected results and some will simply do nothing. E.g. PhotonNetwork.room will, obviously, return null. If you intend on starting a game in singleplayer, but move it to multiplayer at a later stage, you might want to consider hosting a 1 player game instead; this will preserve buffered RPCs and Instantiation calls, whereas offline mode Instantiations will not automatically carry over after Connecting.

Either set PhotonNetwork.offlineMode = false; or Simply call Connect() to stop offline mode.

5.4 Limitations

5.4.1 Views and players

For performance reasons, the PhotonNetwork API supports up to 1000 PhotonViews per player and a maximum of 2,147,483 players (note that this is WAY higher than your hardware can support!). You can easily allow for more PhotonViews per player, at the cost of maximum players. This works as follows:
PhotonViews send out a viewID for every network message. This viewID is an integer and it is composed of the player ID and the player's view ID. The maximum size of an int is 2,147,483,647, divided by our MAX_VIEW_IDS(1000) that allows for over 2 million players, each having 1000 view IDs. As you can see, you can easily increase the player count by reducing the MAX_VIEW_IDS. The other way around, you can give all players more VIEW_IDS at the cost of less maximum players.

It is important to note that most games will never need more than a few view ID's per player (one or two for the character..and that's usually it). If you need much more then you might be doing something wrong! It is extremely inefficient to assign a PhotonView and ID for every bullet that your weapon fires, instead keep track of your fire bullets via the player or weapon's PhotonView.

There is room for improving your bandwidth performance by reducing the int to a short(−32,768, 32,768). By setting MAX_VIEW_IDS to 32 you can then still support 1023 players Search for "//LIMITS NETWORKVIEWS&PLAYERS" for all occurrences of the int viewID. Furthermore, currently the API is not using uint/ushort but only the positive range of the numbers. This is done for simplicity and the usage of viewIDs is not a crucial performance issue for most situations.

5.4.2 Groups and Scoping

The PhotonNetwork plugin does not support real network groups and no scoping yet. While Unity's "scope" feature is not implemented, the network groups are currently implemented purely client side: Any RPC that should be ignored due to grouping, will be discarded after it's received. This way, groups are working but won't save bandwidth.

5.5 Feedback

We are interested in your feedback, as this solution is an ongoing project for us. Let us know if something was too hidden, missing or not working.
To let us know, post in our Forum: forum.exitgames.com

5.6 F.A.Q.

5.6.1 Can I use multiple PhotonViews per GameObject? Why?

Yes this is perfectly fine. You will need multiple PhotonViews if you need to observe 2 or more targets; You can only observe one per PhotonView. For your RPC's you'll only ever need one PhotonView and this can be the same PhotonView that is already observing something. RPC's never clash with an observed target.

5.6.2 Can I use it from Javascript?

To be able to use the Photon classes you'll need to move the Plugins folder to the root of your project.

6 Converting your Unity networking project to Photon

Converting your Unity networking project to Photon can be done in one day. Just to be sure, make a backup of your project, as our automated converter will change your scripts. After this is done, run the converter from the Photon editor window (Window -> Photon Unity Networking -> Converter -> Start). The automatic conversion takes between 30 seconds to 10 minutes, depending on the size of your project and your computers performance. This automatic conversion takes care of the following:

- All NetworkViews are replaced with PhotonViews and the exact same settings. This is applied for all scenes and all prefabs. This should work flawlessly.
- All scripts (JS/BOO/C#) are scanned for Network API calls, and they are replaced with PhotonNetwork calls.

There are some minor differences, therefore you will need to manually fix a few script conversion bugs.
After conversion, you will most likely see some compile errors. You'll have to fix these first. Most common compile errors:

```
PhotonNetwork.RemoveRPCs(player);
PhotonNetwork.DestroyPlayerObjects(player);
     These do not exist, and can be safely removed. Photon
     automatically cleans up players when they leave (even
     though you can disable this and take care of cleanup
     yourself if you want to)
..CloseConnection takes '2' arguments…
     Remove the second, boolean, argument from this call.
PhotonNetwork.GetPing(player);
     GetPing does not take any arguments, you can only request
     the ping to the photon server, not ping to other players.
myPlayerClass.transform.photonView.XXX error
     You will need to convert code like this to:
     myPlayerClass.transform.GetComponent<PhotonView>().XXX
     Inside of scripts, you can use photonView to get the
     attached PhotonView component. However, you cannot call
     this on an external transform directly.
RegisterServer
     There's no more need to register your games to a
     masterserver, Photon does this automatically.
```

You should be able to fix all compile errors in 5-30 minutes. Most errors will originate from main menu/GUI code, related to IPs/Ports/Lobby GUI.

This is where Photon differs most from Unity's solution:

There is only one Photon server and you connect using the room names. Therefore all references to IPs/ports can be removed from your code (usually GUI code). *PhotonNetwork.JoinRoom(string room)* only takes a room argument, you'll need to remove your old IP/port/NAT arguments. If you have been using the "Ultimate Unity networking project" by M2H, you should remove the MultiplayerFunctions class.

Lastly, all old MasterServer calls can be removed. You never need to register servers, and fetching the room list is as easy as calling PhotonNetwork.GetRoomList(). This list is always up to date (no need to fetch/poll etc). Rewriting the room listing can be most work, if your GUI needs to be redone, it might be simpler to write the GUI from scratch.