Easy PUN setup for playmaker

[Update 0.02 – Notes on last page]

Index:

1.Intro
2.PUN setup
A.

With the newer update(s) from plamaker, hutonggames desided to exclude photon and photon FSM's from the package.
You can get the photonplaymaker package from their site.

Installation:
1- import Playmaker.
2- import PhotonPlamaker package.
3- Auto API upgrade if asked.
4- import Easy PUN Setup Playmaker.

B.

To start using this example project or the prefabs we first need to setup Photon/PUN.
Klick on "window" in the toolbar menu and klick on "Photon Unity Networking".
The Photon setup wizard will popup, klick on setup.
Here we have a couple of options, if you dont have a account yet you can submit your e-mail or go the the PUN website to register.
If you already have an account you can setup PUN using your appid or host your own server.
In this case we will setup PUN using a appid.
Klick on setup under "I am already signed up.
Let me enter my AppId" and enter your appid in the "Your AppId" inputfield.

Now chose the region you want PUN to manage your networking trough, after doing that klick save and close the window.

For every scene you want to use over the network you need to add a Playmaker photon proxy.
To do this you go back to the Photon setup wizard, we have a new button to chose.
This button will add the Playmaker photon proxy to your scene.

Now PUN has been setup and you are ready to use the prefabs I have provided you with.

(For a quick start you can use the 2 existing scenes in the project but be sure to read on about the Player to use your own player prefab and instantiate-player.Also make sure that all load level's are changed to fit your game. See Load level num list for a list of states using it.)

3.1 PUNnetworkManager
Location: Your scene where you handle your multiplayer menu's.
Location demo scene: MainMenu.
Folder location: Prefabs.

This will manage your network connection and all the photon global listeners
First it wil connect to the network, while doing this PUNnetworkManager sends feedback to the user.
To give the user feedback showing the network status, we have setup 3 global events for connecting(Orange),disconnected(Red) and connected(Green).
You can use these states to show a visual representation of that state, i left these blank so add your own or just use it as an indicator for the editor.

After PUNnetworkManager has done the connecintg, it wil listen to 2 global events.
Both global events will come from Create&Join rooms Prefab.
CREATED ROOM will be started when a user creates a room in the menu.
JOINED ROOM will start when a user joins a room or after a user creates a room in the menu.
This will load the level from the room the user is joining.
We also disable "Photon Network Set Is Message Queue" or else we wont be able to see players who were already connected before the user connected.
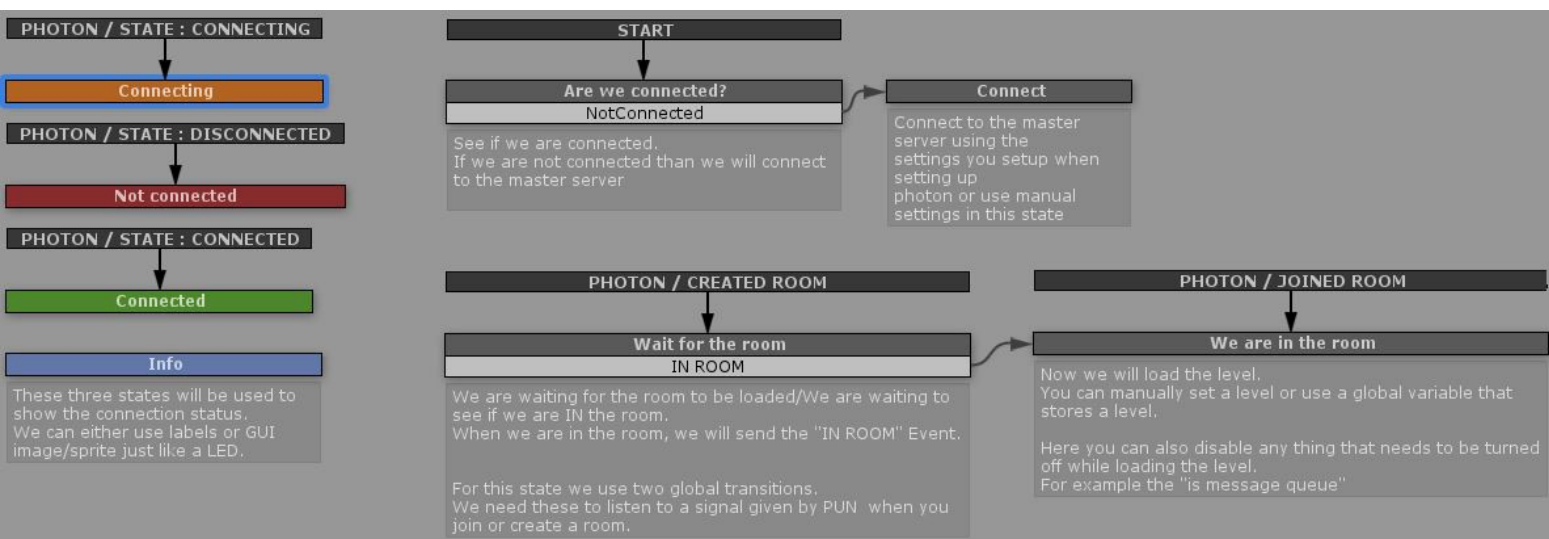
3.2 Create&Join rooms

Location: Your scene where you handle your multiplayer menu's.
Location demo scene: MainMenu.
Folder location: Prefabs

This will manage your joining and creating rooms,this will also send "CREATED ROOM" and "JOINED ROOM " global event to PUNnetworkMaager.
Create&join rooms uses 4   FSM's:

Create room with a name is used to make rooms with names.
This fsm contains a GUI inputfield and button that takes the input from the user and creates a room with it. This GUI will only show if the user is connected,to do that we use the global event "CONNECTED". Also when a user fails to create a room the "CREATE ROOM FAILED" global event will be started. This will show a popup that creating a room failed and wil activate the create room GUI again. If we dont reactivate the create room GUI, it wil not show after failing.
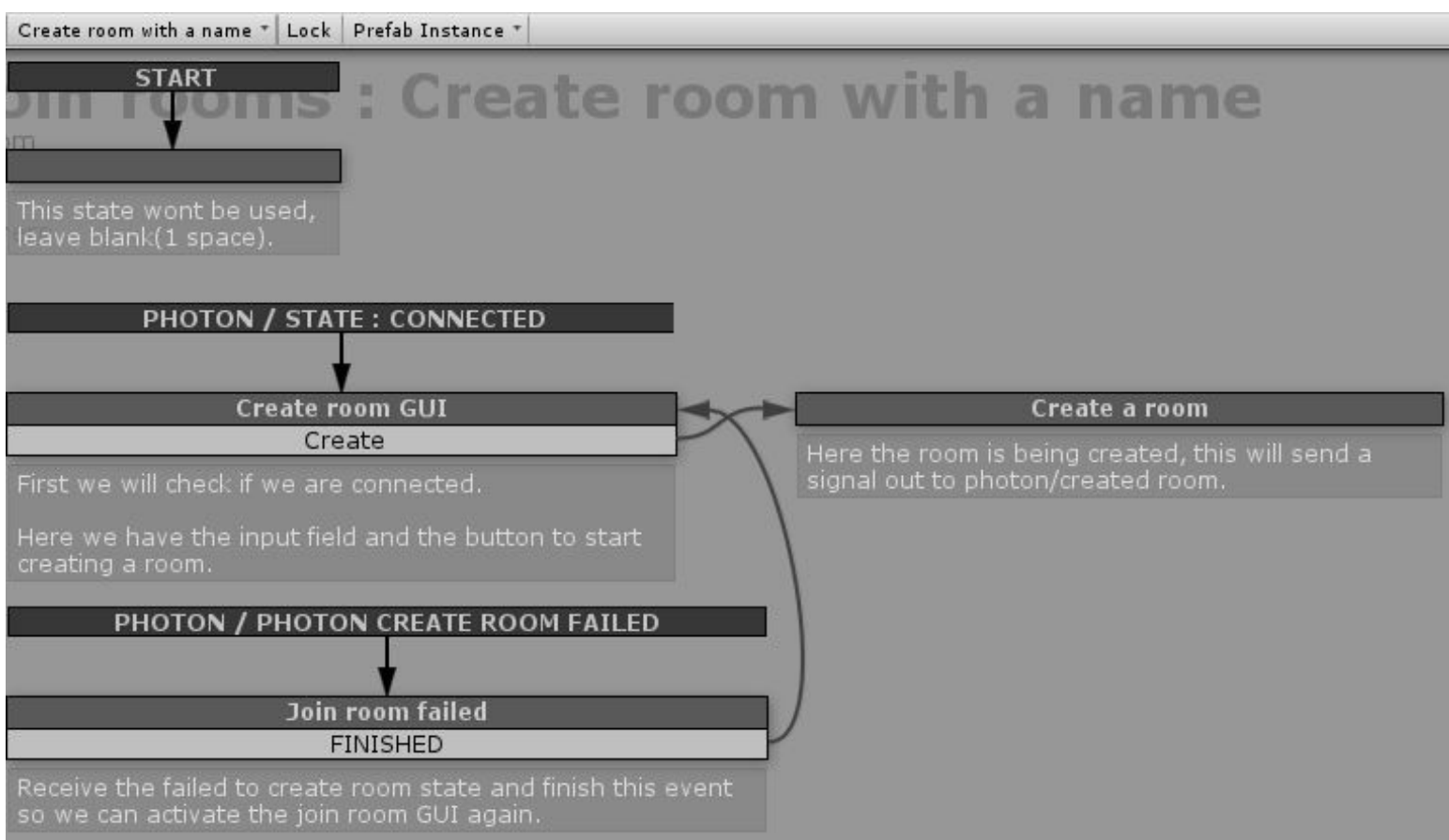
Join room by name is the same as create room with a name but instead of creating a room, the user joins a room based on the name the user inputs in the inputfield.
"JOIN ROOM FAILED" will be activated after a user fails to join a room, this wil reactivate the join room GUI.

Join room by name ▾ | Lock | Prefab Instance ▾

START

This state wont be used, leave blank(1 space).

PHOTON / STATE : CONNECTED

Join room GUI
GO

First we will check if we are connected.

Here we have the input field and the button to start joining a room.

Join a room

Now we will be joining the room, this will send a signal out to photon/joined room.

PHOTON / PHOTON JOIN ROOM FAILED

Join room failed
FINISHED

Receive the failed join room state and finish this event so we can activate the join room GUI again.

Join a random room or create one is used to act as a quick match making system.
This fsm containsa GUI button that lets users join a randome room, a room wil be created if there

Join a random room or create one. ▾ | Lock | Prefab Instance ▾

START

This state wont be used, leave blank(1 space).

PHOTON / STATE : CONNECTED

GUI Button
GO

Here we will make the GUI to start a "Quick Match "The button will send a "GO" event.

Do we have rooms?
Create room
Join room

Here we will look if there are any rooms.

PHOTON / PHOTON RANDOM JOIN FAILED

Create a room

Here you can set the room name and max number of players.

If you leave the room name blank, Photon will chose one for you.

If Join random room fails, this state will activate and create a room.

Join a randoem room

Here you can set the match making mode to three different types.Chose what you like the most.

You can also set the max number of players.
Only set this if its needed, maybe even get the amount from a synced variable.

are no rooms.

To do this the state "do we have rooms?" will be activated after the user pushes the GUI button.

In "do we have rooms?" we will check if there are rooms or not and based on that we will preform the next state.

List of rooms is used to show a list with rooms that the player can klick on and join.

List of rooms listens to 2 global events, these are the "RECIEVED ROOM LIST" and "RECIEVED ROOM LIST UPDATE".

These 2 are used to get the rooms and list them in the GUI and update the list when its changed.

The first state of List of rooms wil check if we have rooms, if there are rooms the state wil activate the list room GUI in the next state.
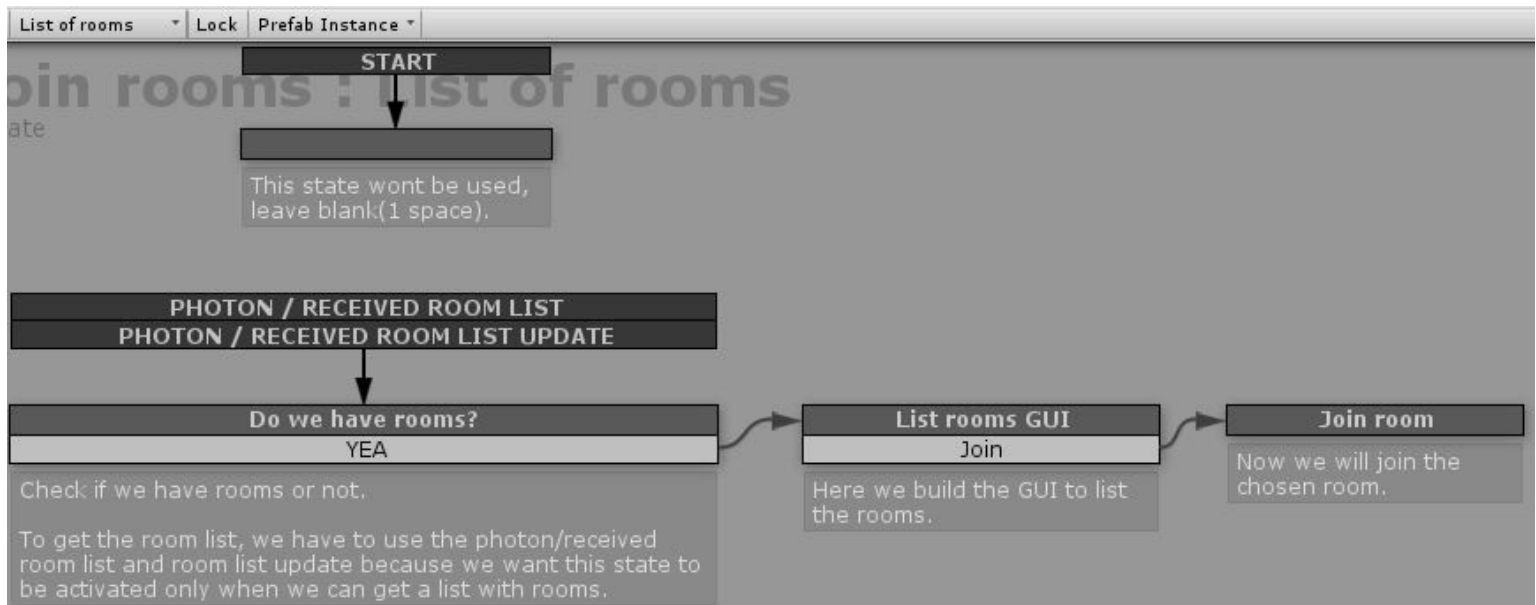
In the list rooms GUI we show the user the list of rooms with a klickable button, if a user klicks on a room the user wil join that room.

3.3 MPplayerOptions

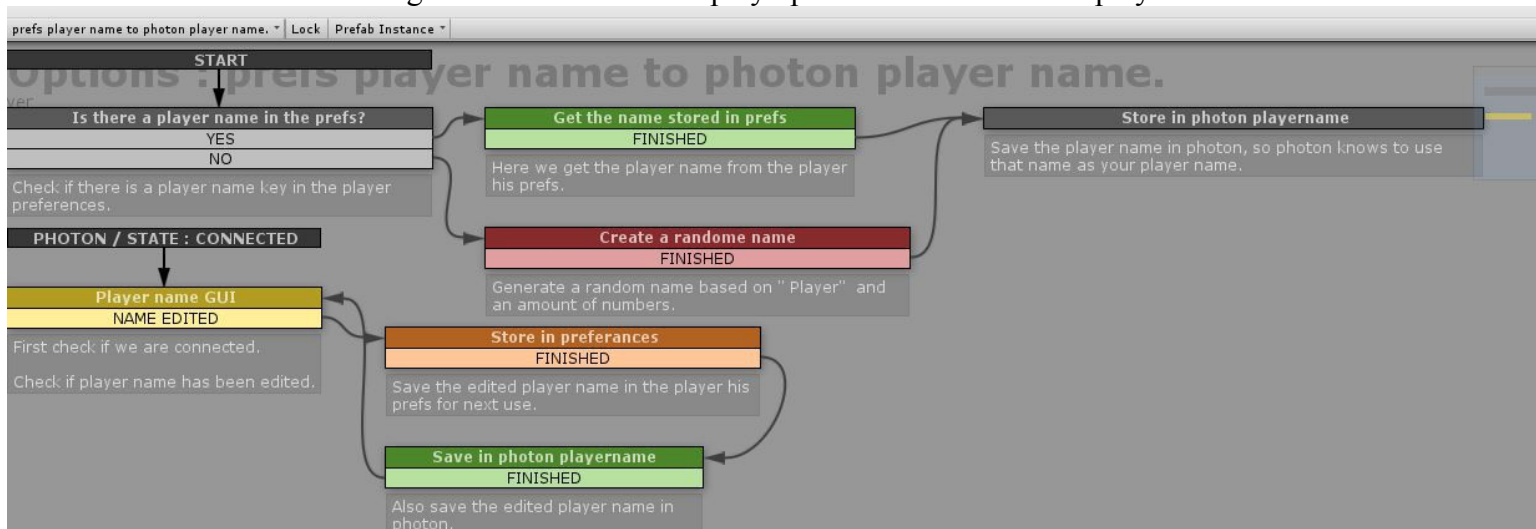Location: Your scene where you handle your multiplayer menu's.

Location demo scene: MainMenu.

Folder location: Prefabs

With this prefab we will handle the player names.

In the fsm there are 2 main states to look at.

The first is the starting state that looks in the playerprefs to see if there is a player name or not.

If there is a player name,it will take it and save it in PUN.
If there is no player name it will make a random name ans save that randome name in PUN.

The second is the global event "CONNECTED" that has the player name GUI.
With this the user can assign his/her own player name in the inputfield.
If the inputfield content has ben changed(using the "changed event" in GUILayout TextField) the next state will be activated, in "store in preferances" the new playername is saved in the player prefs.
When the store in preferances state is finished it will start "Save in photon playername" state to store the new player name in PUN and loop back to Player name GUI.

3.4 Instantiate-player

Location: Every multiplayer scene/level.
Location demo scene: InGame.
Folder location: Prefabs

With this prefab we can instantiate our player in the multiplayer scene/level/game.
At the start of the scene we check if we are connected, we do this to prevent errors and unwanted drawcals and what not.
If we are connected we will enable stuf that has to be enabeled bevore we instantiate the player, what we enable here depends on your game apart from "Photon Network Set Is Message Queue" we always need to enable the is message quing or else we wont see players that are already connected before the user was joining.
Do not enable character related stuf, this is done in Player enabler.

Apart from the starting state we have 3 other states with global events.

"**Reload level**" will be started after the user quits the multiplayer game(left the room) or got disconected from photon, to get this information we use "LEFT ROOM" and "DISCONNECTED FROM PHOTON" global events.
If this state is started the state wil load the multiplayer scene(in the demo scene its level num 0).
To use this state in your own project, make sure you change the load level num to the level index of your multiplayer menu or use load level name instead.

"**Master are we connected**" will be activated when the game gets a new master client.
This means that when the host who created the room leaves, the game wil chose a new host(master) from the users who are connected.
If the game finds a new master the global event "MASTER CLIENT SWITCHED" will be started.
This will start "Master are we connected" so we can check if we are connected to photon.
If we are connected we will contine to the next state to display a message to let users know they got a new master, for example "Hello new master [player name]".
This state will be finished/done in the next update. You can however put your own message in it if you want.

"**Connection Fail**" will show a message to the player with "Error: No connection" and load the main menu after waiting a second.
Change the Load level num level index to your main menu or multiplayer menu, you can use load level name instead of load level num if you want.
This state will only activate/start if "CONNECTION FAIL" and "FAILED TO CONNECT TO PHOTON" global events are started/activated.

4.1 Player
Folder location: Resources

This is our player, it is important to have our player prefab in the folder resources or else photon will not be able to spawn our player.

For this documentation we will only cover multiplayer related FSMs because the controls are not important for networking and will probably be changed to how you see fit for your own project.

Player enabler is where we enable our player related stuf.
This is needed to prevent our selfs and other connected players from controling each other characters.
In this case our player prefab starts with the Lookcontrol, Controlls FSMs and camera disabled.

After the player has been instantiated by the " **Instantiate-player"** FSM, we check to see if this is ours in the "Is Mine?". If its not ours, we will do nothing. If it is ours we proceed to enable the camera in "Enable camera". After the camera is enabled, we will enable the controlls in "Enable

controlls".

You may want to enable other stuf in these states and I recomand you put every thing camera related in its own state(in this case "Enable camera") and all other character/player related in a second state(in this case "Enable controlls").


Sync Position & Sync Rotation are 2 FSMs that will syncronize the users/playsers position and rotation. We will cover this in one subect because both are basicly the same apart from one using Get Position and the other using Get Rotation.

To decide if we need to send or receive the position/rotation we first check if this is our player/character in "IsMine?".
If this is our player/character we proceed to "Get player position"/"Get player rotation" and we will get our position/rotation and store it in a vector 3 variable with the name Position sync/ Rotation sync that we save in the "Synchronized variables" FSM.
If this is not our player/character we proceed to "Set other players position"/"Set other players rotation" where we get the vector3 variable Position sync/ Rotation sync from "Synchronized variables" FSM.
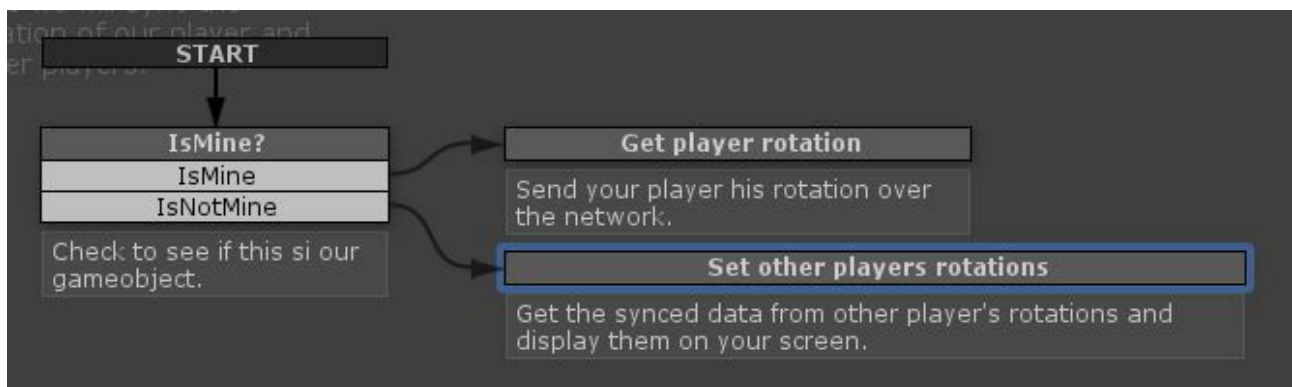
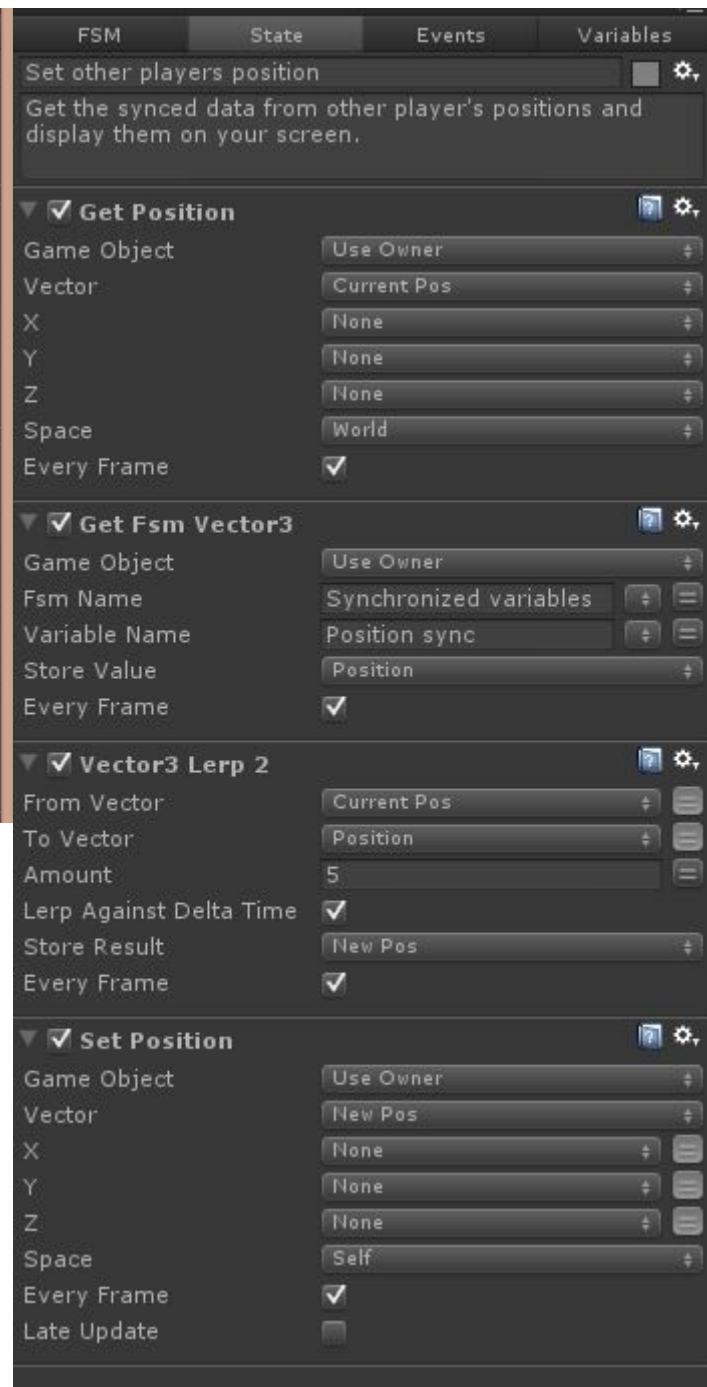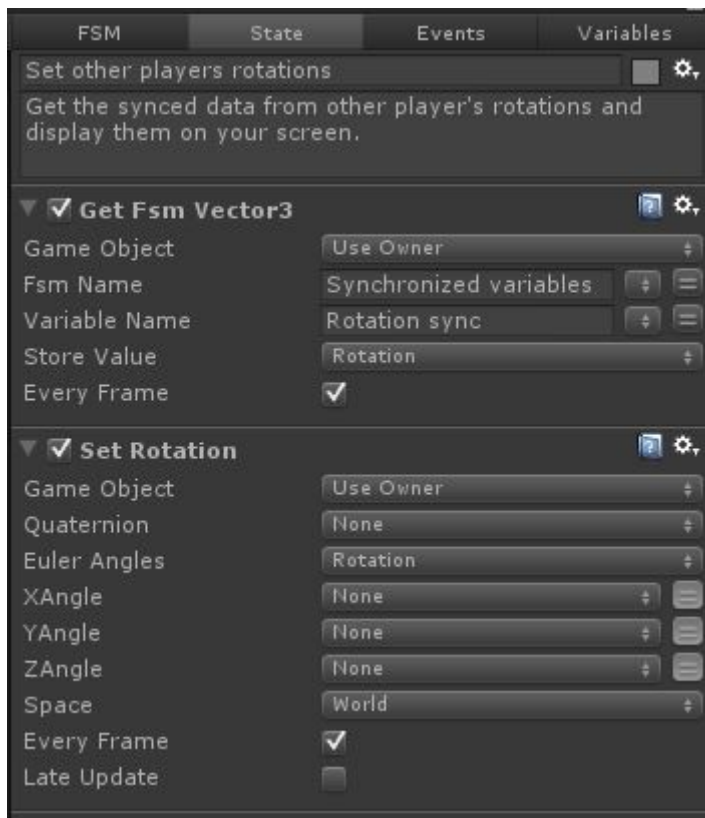However, there is a slight diferance between getting position and getting rotation.
For rotation we only get the variable and do a Set Rotation.
We could do this for position but than the movement wont be smooth and you will see players/users teleport from thair last know position to the new position.
To prefent this from happening we first do a Get Position to save it as Current Pos.
After that we get our vector 3 variable from "Synchronized variables" and save it as Position.
Having both current position and the new position we do a Vector 3 lerp 2 and after that we do the Set Position.

FSM	State	Events	Variables

Set other players rotations

Get the synced data from other player's rotations and display them on your screen.

▼ ☑ Get Fsm Vector3

Game Object	Use Owner
Fsm Name	Synchronized variables
Variable Name	Rotation sync
Store Value	Rotation
Every Frame	☑

▼ ☑ Set Rotation

Game Object	Use Owner
Quaternion	None
Euler Angles	Rotation
XAngle	None
YAngle	None
ZAngle	None
Space	World
Every Frame	☑
Late Update	☐

FSM	State	Events	Variables

Set other players position

Get the synced data from other player's positions and display them on your screen.

▼ ☑ Get Position

Game Object	Use Owner
Vector	Current Pos
X	None
Y	None
Z	None
Space	World
Every Frame	☑

▼ ☑ Get Fsm Vector3

Game Object	Use Owner
Fsm Name	Synchronized variables
Variable Name	Position sync
Store Value	Position
Every Frame	☑

▼ ☑ Vector3 Lerp 2

From Vector	Current Pos
To Vector	Position
Amount	5
Lerp Against Delta Time	☑
Store Result	New Pos
Every Frame	☑

▼ ☑ Set Position

Game Object	Use Owner
Vector	New Pos
X	None
Y	None
Z	None
Space	Self
Every Frame	☑
Late Update	☐

In the player prefab we also sync and enable the "Head" rotation. This works the same way.

There is also a Sync Jumping state that is not being used by the prefab player.
It works the same as syncing the position and rotation and i can asure you that after reading "Sync Position & Sync Rotation" that you will understand how Sync Jumping state works and put it to use or remove it.
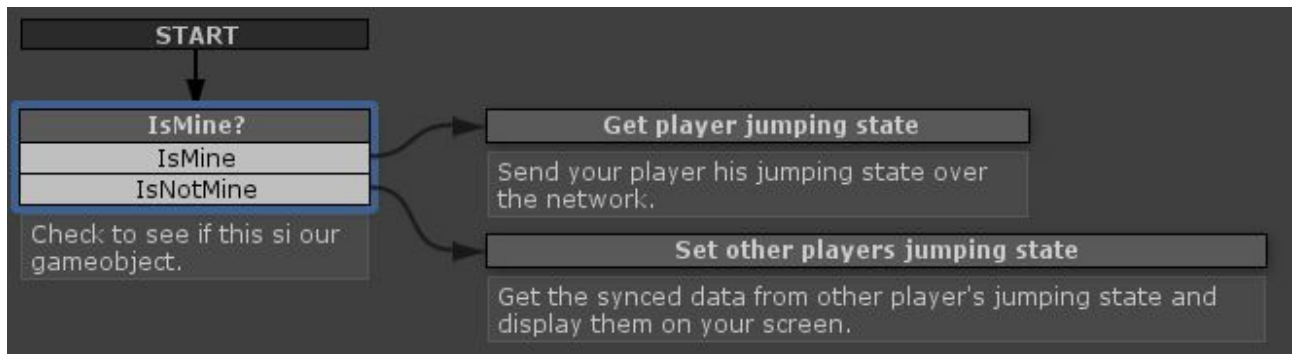

Synchronized variables has no actions and nothing is done here apart from storing the saved variables from other FSMs to sync over network.
To sync a variable you need to create one in the variables tab and make sure that Network Sync is checked, you can ignore the red warning because we dont use a Network view but a Photon View.
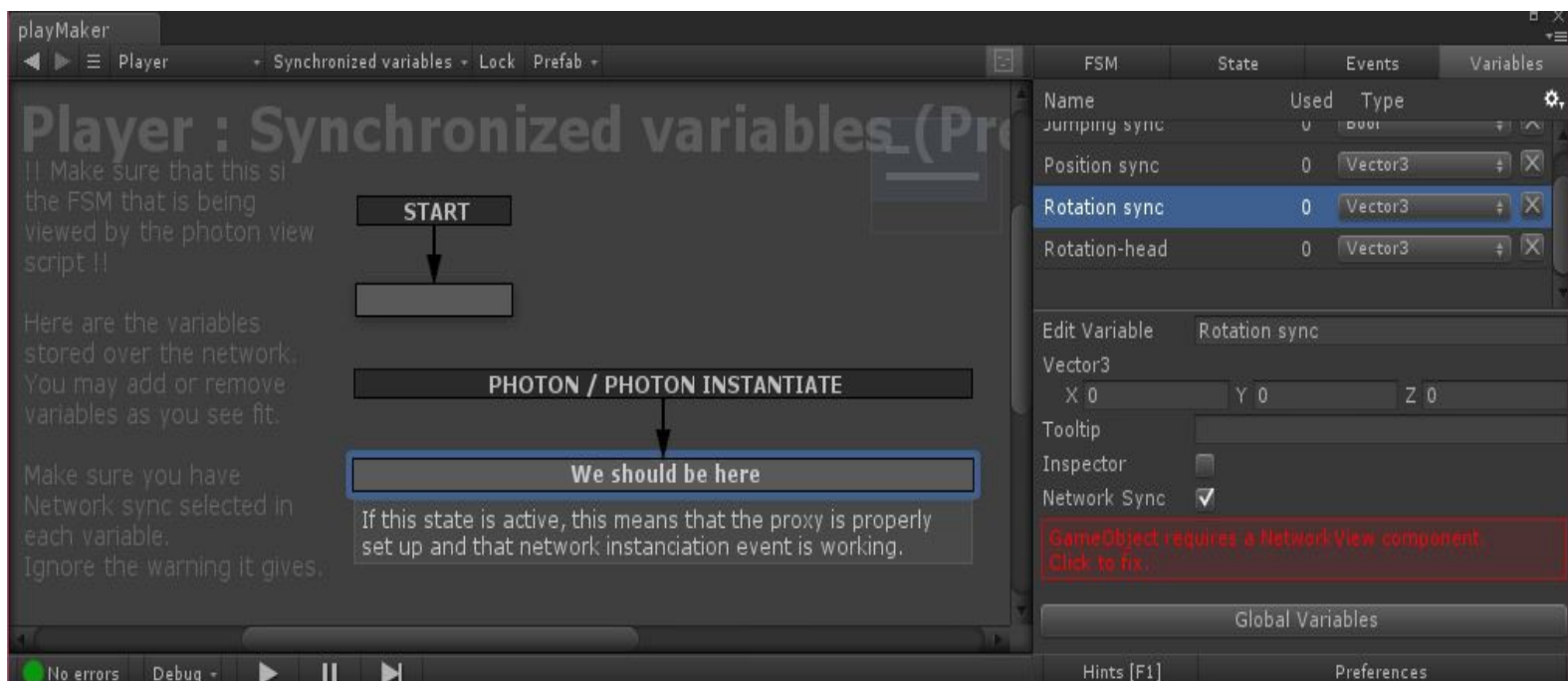
If you make your own Player prefab make sure you ad an extra Photon View component and drag this FSM into the Observe bar. If you dont do this,the Synchronized variables wont be synced over the network.
[See image on next page]
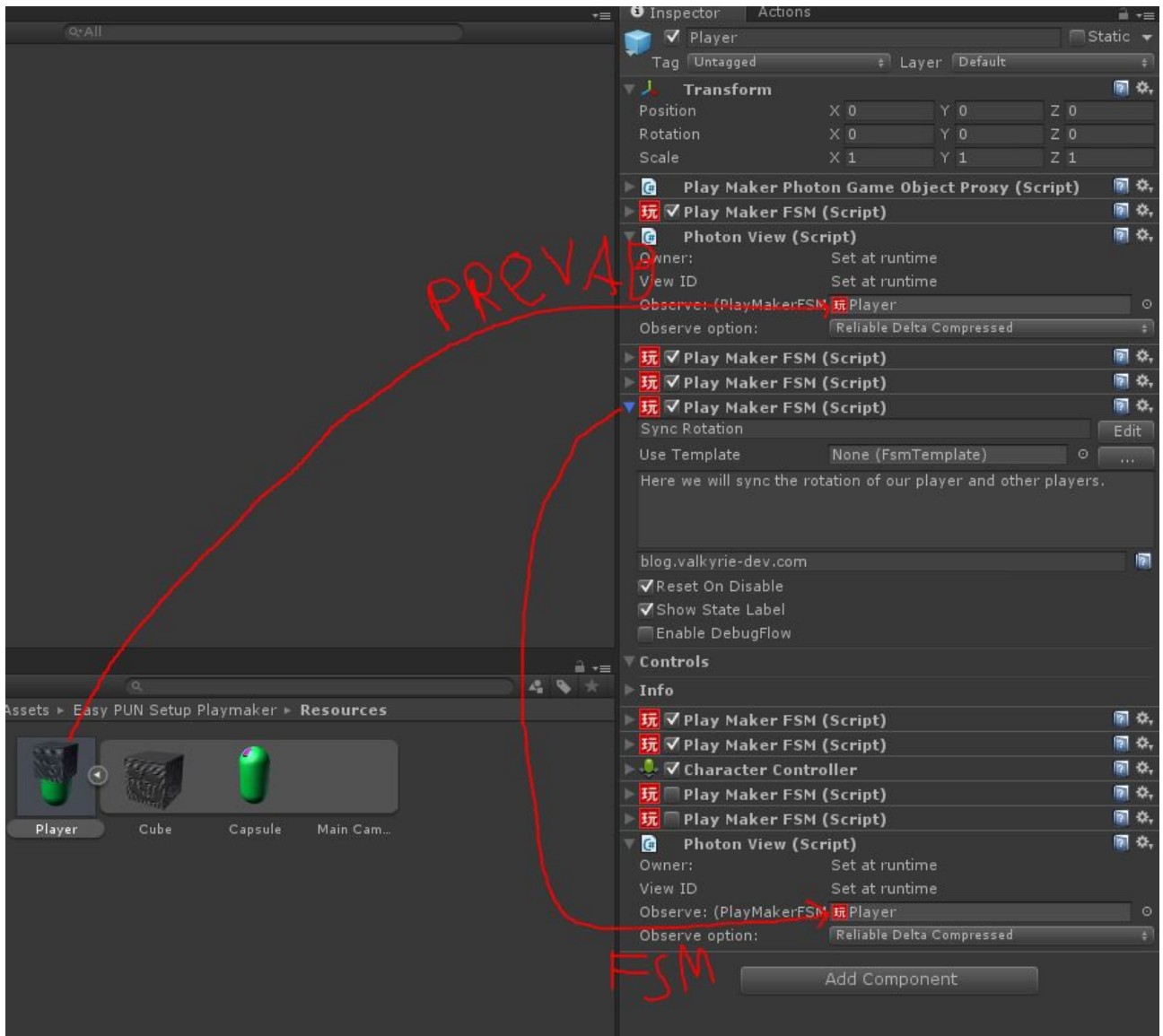
5.1 MainMenu

The MainMenu Scene exists out of:
Prefabs:
*OnlineStatistics
Create&Join rooms
PUNnetworkManager
MpplayerOptions
Photon:
PlayMaker Photon Proxy
Playmaker:
PlaymakerGUI
Scene:
Cube
Directional light
Main Camera
Terrain

5.2 InGame

The InGame Scene exists out of:
Prefabs:
Instantiate-Player
Photon:
PlayMaker Photon Proxy
Playmaker:
PlaymakerGUI
Scene:
Cube
Directional light
Main Camera
Terrain
Capsule
Cylinder
Quad
Sphere

6.Multiplayer setup

Seting up multiplayer is pretty simple with the prefabs in this package.

For the menu you need a scene with:
PlayMaker Photon Proxy
And the prefabs:
Create&Join rooms
PUNnetworkManager
MpplayerOptions

*OnlineStatistics is optional, it shows the player/room/game count and is a dropin solution.
If you dont have any knowladge yet on how playmaker handles GUI I recomand that you follow
some tutorials so you can edit the GUI's from the prefabs.

For the Scenes holding the levels/maps you need:
PlayMaker Photon Proxy
the prefab:
Instantiate-Player
And some spawn locations.

To use more than 1 spawn location, create a empty game object with the name spawn point and give
it the tag Spawn or Spawnpoint.
Place them any where you like.

Now to use these spawnpoints we need to edit the Instantiate-Player prefab.
Once in the FSM window, go to the state "Instantiate the player" and add the action Get Random
Object bevore/in front of Photon Network Instantiate.
Set the tag in Get Random Object and store the result in a gameobject variable, create one and call it
Spawnpoint.
Now in Photon Network Instantiate set the Spawn Point to the variable and you are done.

7.Load Level num list
These are the FSMs with the Load level num actions.
Change these to make it work with your scenes in your game.

PUNnetworkManager
  We are in the room

Instantiate-player
  Reload level
  Connection Fai

8.1.1. Practise of shooting

With this topic I want to cover a couple of ways to synchronize your shooting actions over the network.

Fast projectiles(Bullets,Lasers, ect):
The best way to do this is probably to not sync this at all.
Now this may sound strange to you but I will explain in to you.
Bullets mostly exists out of gameobjects or "......" and go really fast. For the server to sync this, it needs to sync every time a bullet is created,the way it go's, how fast ect. With a automatic rifle this could hog up the network pretty fast or players might just not get the hits they want due to lag or slow networking.

So what do we do instead? Make the local instances of players do a shoot animation and some fake rays if its at taste. So we basically only need to sync the players direction its facing and a stat that he is shooting. This will make the local instance of the player do the same on a other player his machine.

Now there is still a possibility that on one screen of both players, it never hits the target, thats why we only animate.
To solve this we have 2 options.
Either calculate the damage on the shooting side or calculate it on the target side.
Most games do the first, bullets wont do damage on target side but the damage being done on shooting side will be synced to the target side(this prevents double damage).

Slow projectiles(Rockets,NoobTubes ,ect):(Will be added later)
Guided projectiles(Controled rockets, ect):(Will be added later)

Now have some multiplayer fun.

Update 0.01:

Prefab Instantiate-PLayer:
1. FSM [For update] is now "Player counter&Action".
   1. *Added descriptions and updated states.*
2. Updated FSM"Instantiate" with new descriptions.
   1. *Renamed 2 states and moved 1 global event.*

Update 0.02:

1. Fixed an fsm that made the player quit the room while creating one.
2. Added a install description to the documentation.
3. Removed Terrain in "MainMenu" and added a new terrain in "InGame", the old terrain was missing.