

Exploration on Multiplayer Networking Capabilities in Unity using Photon Unity Networking Cloud API

Muhammad Reifiza

School of Electronics Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13514103@std.stei.itb.ac.id

Abstract—Multiplayer game is massively popular today. With the availability of modern game engine such as Unity, it's easier to develop a game than 15 years ago. The challenge to create a proper and engaging multiplayer still exist, but at least in Unity there are quite plenty of tools to aid development of multiplayer game. In this paper, an exploration of an existing tool for making development multiplayer game easier without worrying about building complex networking system will be presented and discussed.

Keywords—multiplayer; game; development; api; exploration; unity; photon

I. INTRODUCTION

Nowadays, multiplayer games is massively and increasingly popular worldwide. Branches of multiplayer games also incredibly popular and some have tremendous amount of daily active users. Notable example includes Defense of The Ancient 2, World Of Warcraft, and League of Legends. Multiplayer games are thought to be more engaging to the player than single player games because player is socializing to other players, have competition factor in it, and overall made achievement accomplishment more satisfying and made players more immersed as they play [1,2,3]. These led to another conclusion that multiplayer aspect of multiplayer games more likely to play much more important role than any other aspects, thus driving player to willingly learn complex gameplay mechanics, enhance player persistence and enjoyment, and further might spending dollars in order to win [1,3]. Hence, developing and marketing a multiplayer game seems to be more tempting than developing and marketing a single player game in today gaming industry.

However, developing a software architecture for multiplayer games from scratch without the use of game engine and additional tools could be incredibly challenging and more expensive. There are plenty aspects to be carefully designed and implemented. For example, in order to create a simple multiplayer games, there must be a host and a mechanism to control the players. The mechanism includes controlling state of players, state of network, data flow in gameplay, et cetera. Scalability concerns could present, as well as network and security problems. Network problem includes dealing with network latency might be high, reliability might be not so reliable, limited bandwidth,

etc. Tackling these problems might be more of headache and there could be other problems not mentioned yet.

Fortunately, there are exist game engines and tools to make development of multiplayer game much easier. For this exploration project, I'll be using Unity as game engine. There are at least two APIs to develop multiplayer game in Unity, built-in UNet API which requires Unity Multiplayer Services and Photon Unity Networking which has existed before Unet API. As the title says, I'll use the latter for this exploration project.

This paper is divided into four section. Section I explains motivation behind the exploration project and brief introduction, Section II will briefly explain PUN Cloud API basics, Section III will explain how I explore the PUN Cloud API, and the last section will show the result of exploration and conclude the exploration project.

II. PUN CLOUD API BASICS

According to its official documentation, Photon is a real-time multiplayer game development framework that is fast, lean and flexible [4]. The PUN Cloud API consist of two main parts, one located in the client side, and another located in PUN-owned servers. The source code of client side API can be accessed and easily readable by developers, but source code of the server side can't be accessed, making it somewhat half open source.

Photon Unity Networking API is intended for realtime room-based multiplayer game. This means many players can connect to PUN servers, but are separated based on room, application ID, and application version. Players that are in the same room can communicate to each other, but players in the different rooms can't communicate. A room can be set to a limited number of player and its visibility can be set to public or private. If a room visibility is set to private, then it can't be searchable, thus player who want to join in must know the room name. A collection of rooms resides in a lobby, and an application can have multiple lobbies in PUN Cloud Server. All of these information is managed in PUN Cloud Server, hence matchmaking also takes place in server side of API.

III. THE EXPLORATION PROCESS

This exploration project is based on Photon Unity Networking API official documentation and tutorial. Tutorial used mainly is tutorial PUN Basics [5]. Along the exploration, I retyped the original code, had changed several parts of the code, experimenting with other methods, figuring out what's wrong, before changed it again to its original.

The exploration project is very simple. Player can control an anthropomorphic robot with W/ArrowUp to run north, A/ArrowLeft to run east, and D/ArrowRight to run west. The robot can jump while it runs, but not when it is in idle position. The robot can shoot laser beams and explore an arena with predetermined size. Another robot may join into or out from the same arena and the arena will adapt its size according to number of the robots. If a robot is in contact with laser beam, then its health is reduced. An arena can contain up to 4 robots at once.

Before a player can join an arena, the player must first enter a desired nickname. This nickname is stored in local machine registry key, and will be remembered next time the player launches the game. The original code will take the player into an arbitrary room available and if there's no available room, a new room will be created. It's actually possible to not take the player into an arbitrary room. First, we must retrieve all of available rooms with visibility set to public, display it on the player machine nicely so the player can then select any of it, or if the player knows a room name, just provide a form field to let player enter it and if the room really exists then the player will enter the room. All of these approaches are pretty straightforward, but one thing to keep in mind is once all players have left the room, it will be destroyed and won't exist anymore.

Once gameplay has started, every player can see other players in the arena and every player is only able to control its own robot. Every player's robot is attached to a PhotonView class and instantiated on the network by Photon by calling method PhotonNetwork.Instantiate(arguments). What this method does is player's robot is spawned on every player's machine connected to the same room, distinguished by its id. So, if there are two players in the room, then every machine has exactly two instances of robots. Hence, if not handled properly, a player can control all robots and every player can do the same, leading to a catastrophic situation. PUN Cloud API has an ownership concept to avoid this situation that is simpler to grasp than ownership in UNet API, because in Photon there's only two: `isMine` and `isMasterClient`. I use the `isMine` attribute to avoid the situation, as well as in original code. `isMine` attribute is used to check whether a robot instance is not originated from the network, while `isMasterClient` is used to check whether a player is the one who created the room or the one who is in the room for the longest time.

Every player can also see other players' robot's movements, health, and laser beams, thus introducing the needs of data synchronization. There are at least 4 ways to achieve this: Photon built-in script; serializing PhotonView via method callback; Remote Procedure Calls; and RaiseEvents. Because robot movement data is challenging to handle manually, Photon built-in scripts are used in collaboration with other scripts. Health and laser beams are only primitive data types, so

serializing PhotonView option is used. Using Remote Procedure Calls and RaiseEvents isn't the best option though as it only added unnecessary complexity because all the data that needs to be synchronized is the same across the network, and each data is updated rather frequently.

After several attempts of debugging, the game is deployed into a Windows PC with a working internet connection, then multiple instances of the game are executed in parallel.

IV. CONCLUSION

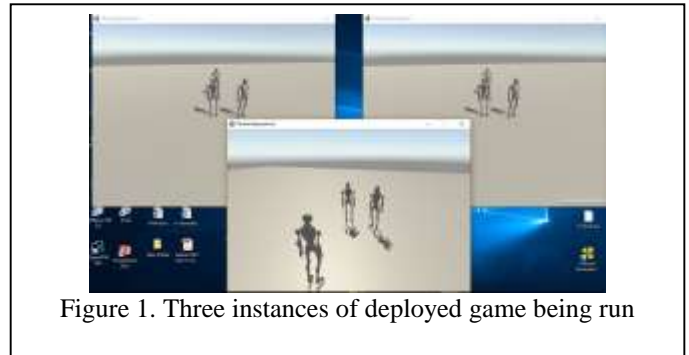


Figure 1. Three instances of deployed game being run

The game works as expected. Player in upper-left window is only able to control one instance of robot, as well as player in other windows. Every player can see other's robot and its movement. The arena is adapting its size based on the number of players. Overall works great, but there's a small notable lag occurred.

PUN Cloud API offers a straight-forward and easy-to-use tools in combination with Unity Engine to develop a multiplayer game. There are several differences between PUN Cloud API and the built-in Unity Networking API (UNet API). One notable difference is PUN Cloud API is always connected to a dedicated server whereas UNet API uses one of player's machine to host a room for multiplayer game, thus if the player's machine used as host crashes then the game is lost too. Both APIs don't offer any way to store persistent data in their servers and developers still need a dedicated host server to store persistent data.

ACKNOWLEDGMENT

I thank to Allah S.W.T. for giving me health and chances to complete this task, and I also being grateful to all my lecturers, especially Mr. Rinaldi, Mrs. Ayu, and Mrs. Dessi. Thank you.

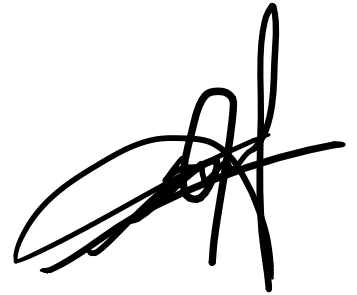
REFERENCES

- [1] Boyle, A.E., Connolly, M.T., Hainey, T., & Boyle, M.J. "Engagement in digital entertainment games: A systematic review" in *Computers in Human Behavior*, vol. 28. Elsevier, 2012, pp. 771-780
- [2] Preece, J., Rogers, Y., Sharp, H. *Interaction Design: Beyond Human-Computer Interaction* Fourth Edition. Chichester: John Wiley & Sons, 2015
- [3] Kellar, M., Watters, C., Duffy, J. *Motivational factors in game play in two user Groups*. 2005
- [4] <http://doc-api.photonengine.com/en/pun/current/>
- [5] <https://doc.photonengine.com/en-us/pun/current/tutorials/pun-basics-tutorial/>

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Mei 2017

A handwritten signature in black ink, consisting of several overlapping loops and a long horizontal stroke extending to the right.

Muhammad Reifiza / 13514103