Simple Online FPS

Oleg Loshkin

January 15, 2020

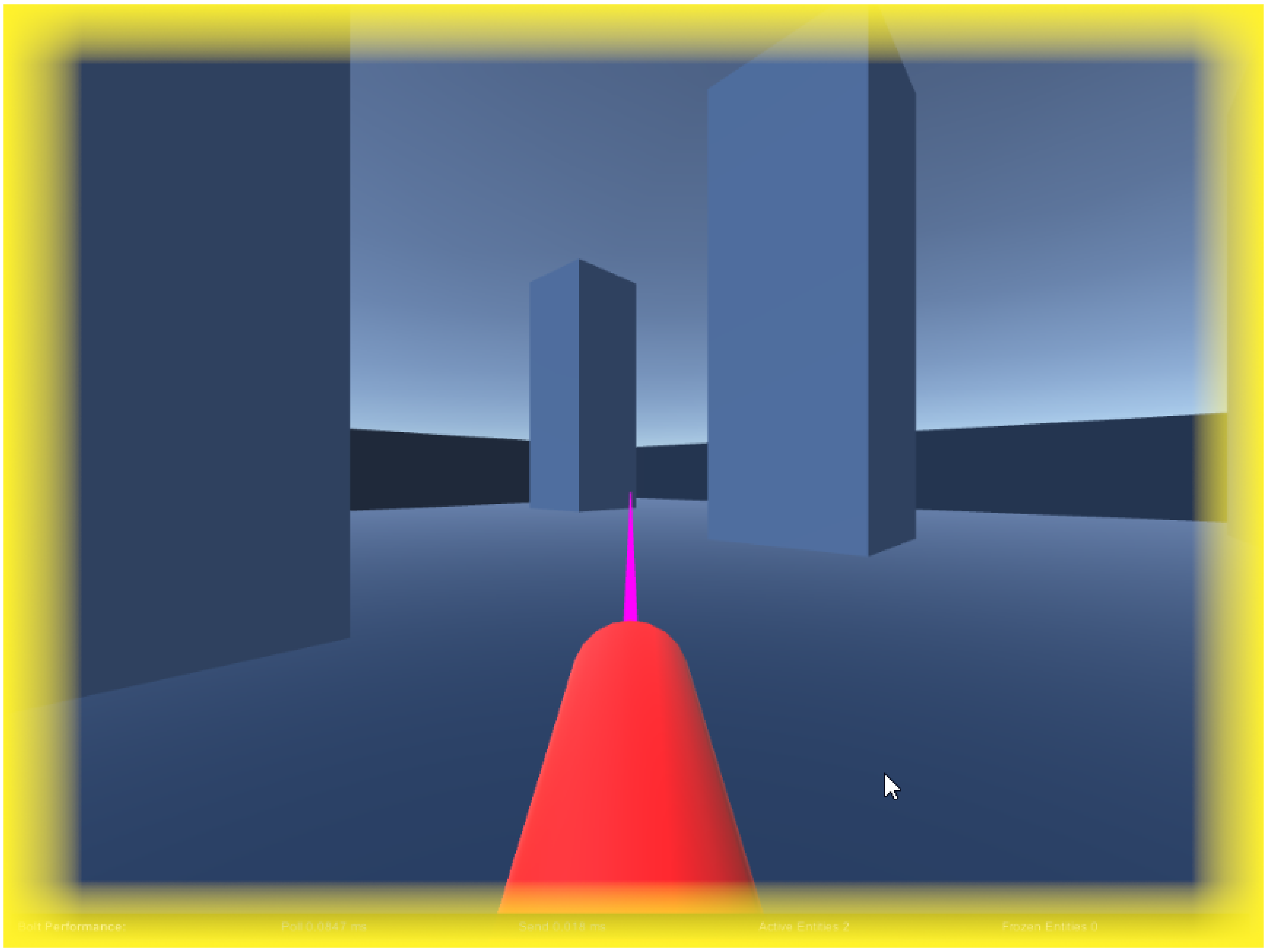
1 Introduction

This game is the result of the **SAE's repeat session for the GPR5100 Game Networking module**. **The aim of this project was to make a simple but functional online game** using Unity and a networking technology of our choice.

2 Technologies used

- **Unity 2018.4.9f1:**
  Latest available LTS version of Unity available at the time the project had started.
- **Photon Bolt:**
  High level game networking API for Unity. **The API provides some features that match perfectly the requirements of an online FPS**, namely lag compensating hitboxes and ray-casting which negates the need to implement this complex feature manually.
- **Github:**
  Github was used for versioning. **The project's repository is available** here .
  The repository was temporarely split between two branches mid project to explore a possible migration to Photon Pun. This idea was abandonned and game was implemented using Photon Bolt.

3 Gameplay



- **The Lobby:**
  The player can either **create a new game session** and name it **or join a random game session currently in progress**. Upon the creation of a session, the host joins it as a listen server.
  Note that the session's name is currently unused as no list of sessions is visible anywhere, that sessions cannot be password protected and the number of maximum players per session cannot be established.

- **The Arena:**
  The arena is a **square area surrounded by walls and dotted with pillars that server as cover**. **Spawn points are located on the outer periphery of the arena close to the pillars**.

- **The Controls:**
  Players can **shoot with Left Mouse Button**, **look around horizontally only using the mouse** and **move around using WASD keys**.

- **The Fight:**
  A 2 second cooldown is enforced between each shot a player takes. **A player can take two hits from an adversary before dying**. No ways to regenerate health exist. **Upon death, the player respawns at the spawn point furthest away from the average position of all other players**.

- **The Feedbacks:**
  - A **shooting sound** is played upon any shot.
  - A **reloading sound** is played just before the end of the shooting cooldown and **the sound's end indicates the end of the cooldown**. Only the local player hears this sound.
  - **Getting hit** triggers a **grunting** sound from the player. Only the player being hit hears this sound.
  - **Upon landing a shot** on a player, **a ringing sound is played** for every player except the one being hit.
  - Upon **being hit once**, the player's **screen edges turn yellow**. **Being hit a second time** turns the **edges turn red**. A third hit kills the player and the health indicating overlay resets.

4 Movement

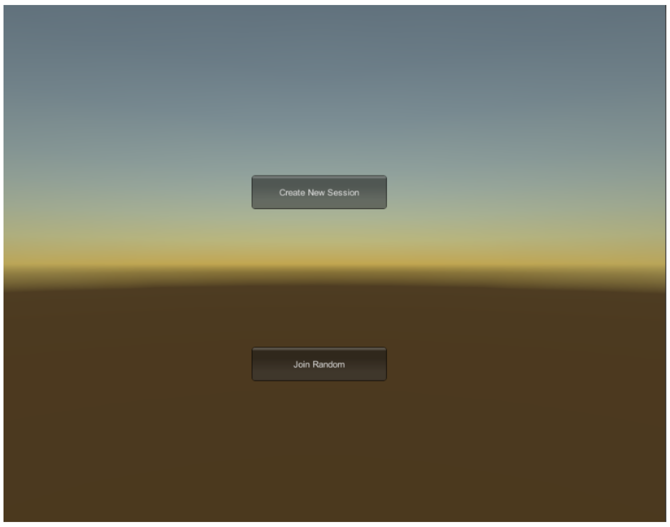
The movement in the game is **client authoritative** and uses **interpolation**. It is implemented using **Photon Bolt's Transform State and Commands**. While using RigidBodies is impossible in conjunction with Photon Bolt, Unity's "Character Controller" component has provided a good replacement for movement and collisions.

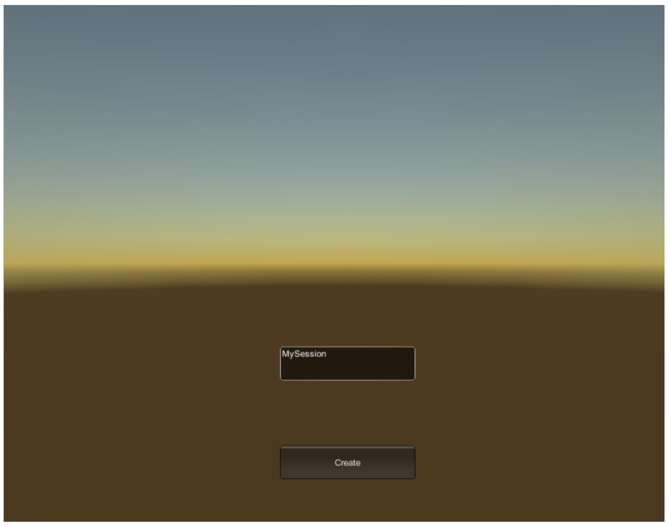
Client authoritativity not being ideal in online FPS games, a server authoritative movement was originally planned. However, after much efforts trying to make it work, Photon Bolt's rollback and lack of documentation of it's inner workings has made this idea too difficult to execute.

5 Hitscanning

Implementation of hitscanning was surprisingly easy using **Photon Bolt's Hitboxes and it's BoltNetwork.RaycastAll() method**. The only caveats that had to be worked around was the inability to filter the targets and the fact that the raycast does not interact with non-Bolt game objects like walls.

6 Lobby





The lobby is comprised of **two tabs**: a main one with two buttons for **creating a new session or to join a random one**, and **a tab for naming the new session before joining it**.
A different lobby was originally planned, **one that would display a list of all current game sessions** in progress with their name, **the number of players** and whether or not they are **protected by a password**. **This could not be implemented for the following reasons:**

Photon **Bolt provides a static property under "BoltNetwork.Sessions"** to allow the program to retrieve all ongoing sessions. **However, this list of sessions is unexplicablely always empty**, even for sample projects provided by Bolt. Even after much effort trying to resolve this bug, **no way was found to fix it**, the idea of **displaying a list of ongoing sessions had therefore to be abandoned**.
Instead, Photon Bolt's **BoltMatchmaking.JoinRandom()** static method **is used to join any ongoing game session, bypassing the sessions list completely** and successfully joining any existing sessions.
Since there is no way to retrieve a list of any ongoing sessions to check if any exist, **clicking the "Join Random" button** in the UI when there are **no sessions running**, **locks the player indefinitely in a connecting state** and the player has to restart the program to exit it.
A **password protection and maximal number of players per session** limit was **not implemented** either **due to the same bug**: there is **no way to retrieve any data about the session** if the session list is never updated.

For these reasons, the lobby is as minimalistic as it is.

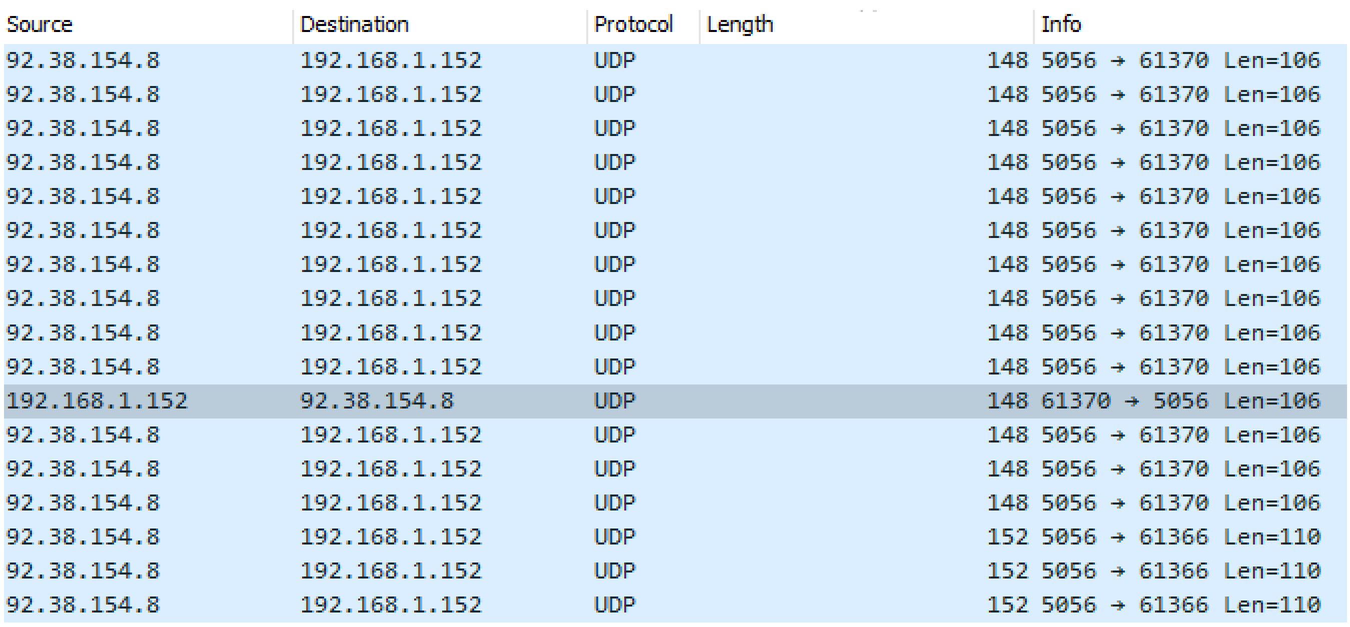
7 Gameplay Feedbacks

Feedbacks for the gameplay have been implemented using **Photon Bolt's Events**. This allows the various feedbacks to be played on the clients of all players in a session when needed.

8 Bot Behaviour

A player can **press the "1" key** on the keyboard to **enable/disable a simple automatic back and forth movement**. This may be used to test latency / lag compensation.

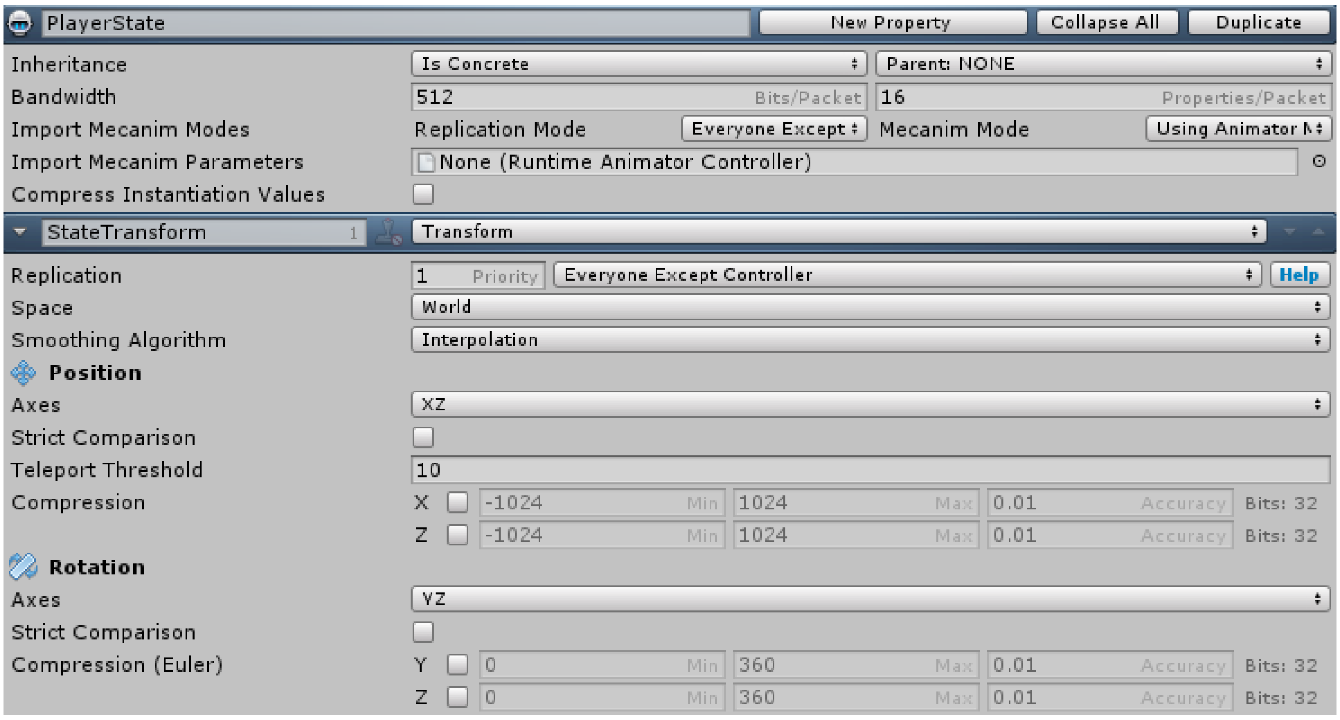
9 The Packets



Source	Destination	Protocol	Length	Info
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
192.168.1.152	92.38.154.8	UDP	148	61370 → 5056 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	148	5056 → 61370 Len=106
92.38.154.8	192.168.1.152	UDP	152	5056 → 61366 Len=110
92.38.154.8	192.168.1.152	UDP	152	5056 → 61366 Len=110
92.38.154.8	192.168.1.152	UDP	152	5056 → 61366 Len=110

Photon **Bolt sends and receives merged UDP packets** at a fixed rate defined in the framework's settings. For this game, it sends **25 UDP packets / second.**
The **packet's size varies between 60 to 200 bytes** in total. The **ports used** for these packets are in the range of **5055 to 27002**.

The **traffic is minimized by sending only the relevant data** which is defined via the Photon Bolt's State editor. **In the example below**, you can see that **only the value of relevant axes are sent** over the network for the position and rotation of the PlayerState for instance.
Moreover, Photon Bolt sends these values only if they have changed, further reducing the packet's size if no changes were made to the PlayerState between two game frames.



```
PlayerState                                  New Property   Collapse All   Duplicate
Inheritance                    Is Concrete                    Parent: NONE
Bandwidth                      512             Bits/Packet    16              Properties/Packet
Import Mecanim Modes           Replication Mode  Everyone Except   Mecanim Mode   Using Animator M
Import Mecanim Parameters      None (Runtime Animator Controller)
Compress Instantiation Values  ☐
StateTransform               1 Transform
Replication                    1 Priority  Everyone Except Controller          Help
Space                          World
Smoothing Algorithm            Interpolation
Position
Axes                           XZ
Strict Comparison              ☐
Teleport Threshold             10
Compression                    X ☐ -1024 Min  1024 Max  0.01 Accuracy  Bits: 32
                               Z ☐ -1024 Min  1024 Max  0.01 Accuracy  Bits: 32
Rotation
Axes                           YZ
Strict Comparison              ☐
Compression (Euler)            Y ☐ 0 Min  360 Max  0.01 Accuracy  Bits: 32
                               Z ☐ 0 Min  360 Max  0.01 Accuracy  Bits: 32
```

10 Potential Improvements

- **Clicking "Join Random" in the lobby without any running sessions locks the client into an indefinite state** where they can't do anything and that they cannot escape without restarting the game. This is due to the bug described in the Lobby section that prevents the implementation of a check to verify the non existence of any sessions.
- **While the player can name a session when creating one**, due to the above mentioned "BoltNetwork.Sessions" bug, no session list are displayed in the lobby and the name given by the player upon session creation **is not used** in any way.
- Sessions cannot currently be protected by a password due to the "BoltNetwork.Sessions" bug.
- In approaching the architecture of the program, I failed to consider the complexity that a multiplayer game implies. Therefore, while the **current architecture** gets the job done, it **would certainly benefit from some serious refactoring** to make it more granular, meaning to make the program very explicitly take into account that each player has their own worlds and that these need to be synchronized.

11 Summary

The game fulfils it's basic ambitions: **it is a barebone online FPS that successfully synchronizes players, performs hitscans for firing and triggers the appropriate gameplay feedbacks across the clients of all players.**

Photon Bolt has proven to be a difficult framework to work with. While it does provide some very useful features like the the synchronized hitboxes, it's transparency both in the code and in it's documentation makes debugging very difficult or even impossible like is the case with the "BoltNetwork.Sessions" list bug.
As a result, much of the time that has been allocated to this project was spent trying to solve unexplicable bugs or finding workarounds instead of expanding the features of the game, and some crucial features were outright impossible to implement.

In conclusion, I am quite satisfied with the end result, as meagre it may appear. During this project I've acquired the experience of working with a rather uncooperative framework, implementing the rather complex task of a responsive real-time online game, while tackling the problems encountered in a way to still provide a viable game at the end of the project.