# 프로젝트 보고서

네트워크프로그래밍(가)반

프로젝트 명 : Footprint

1 조 김건학(20192804), 안아영(20201737), 이경호(20192854)

# 목차

# 1. 서론

## 1.1 프로젝트 목적

이 프로젝트는 오픈 월드형 게임을 대상으로, 게임 플레이 중에 발생하는 패킷 데이터를 분석하고 이를 활용하여 오픈 월드 게임에서 필수적인 기능을 제공하는 어플리케이션을 Window 운영 체제에서 개발하는 것을 목표로 한다. 오픈 월드 게임은 사용자가 가상 세계를 탐험하고 자유롭게 오브젝트에 접근하며 게임 플레이를 즐길 수 있는 게임 유형으로, 이 프로젝트에서는 해당 게임에서 필수적인 좌표 값, 이동 등의 데이터를 추출하고 분석하여 플레이어에게 미니맵의 기능을 제공할 예정이다.

본 프로젝트에서는 오픈 월드 게임의 특징을 가지고 있으며, 데이터 추출에 용이한 암호화되지 않은 패킷 전송 방식과 UDP 프로토콜을 사용하고 있는 게임인 Stardew Valley의 좌표, 이동 데이터와 해당 데이터들을 가지고 있는 패킷들을 분석한다.

## 1.2 프로젝트 필요성 개요

Stardew Valley 와 같은 오픈 월드형 게임에서 발생하는 패킷 데이터를 분석하여 유용한 기능을 제공하는 어플리케이션을 개발하여 게임 플레이어들이 더 원활하고 향상된 게임 경험을 제공한다.

구체적으로, 해당 게임에서 추출한 패킷 데이터를 분석하여 오픈 월드형 게임이 필히 가지고 있는 데이터인 좌표값, 맵 정보, 이동 경로 등의 유용한 정보를 활용할 것이며 이러한 데이터를 기반으로 미니맵 기능을 구현하여 플레이어가 자신의 위치를 시각적으로 파악하고, 모든 플레이어들의 좌표를 실시간으로 확인할 수 있다. 이는 Stardew Valley 와 같이 미니맵 기능이 존재하지 않는 오픈 월드형 게임에서의 한계를 극복하고, 게임의 깊은 탐험과 상호작용을 더욱 가능하게 만들어 플레이어들에게 흥미로운 게임 경험을 제공하는 것을 목표로 한다.

# 2. 프로젝트 분석

## 2.1 프로젝트 필요성 및 특장점

오픈월드 게임의 가장 큰 특성인 이동의 자유로움은 플레이어가 맵 내에서 동선의 제한 없이 플레이할 수 있게 해준다. 이는 게임의 자유도를 느끼게 해주지만 이러한 특성을 이용하여 게임적 요소를 풀어나가기 때문에 오픈월드형 게임에서의 위치를 알 수 있는 좌표값은 꽤나 중요하게 작용한다. 특히, 다수의 플레이어들과 함께 상호작용하며 플레이하는 경우 플레이어들 간의 위치 공유가 원활하게 이루어져야 유연한 플레이가 가능하다. 그러나 이러한 기능을 제공하는 것은 몇몇 게임에 해당하지 않아 플레이어로 하여금 불편함을 유발하게 하는 경우가 많다. 이러한 점에서 착안한 Footprint 는 오픈월드형 게임에서 자신의 위치를 파악하고 각 플레이어들의 위치(좌표), 원하는 플레이어의 위치 정보들을 미니맵을 통해 공유함으로써 다른 플레이어와 상호작용하는것을 도와주는 기능을 제공하고자 한다.

네트워크 패킷 캡쳐 및 분석 소프트웨어인 Wireshark 를 활용해 플레이어간의 주고받은 패킷을 분석하여 얻은 유저의 플레이 정보(유저의 경험)를 기반으로 게임 내에서 지원하지 않는 기능을 새롭게 만들어 추가적으로 제공함으로써 게임 유저들에게 기존 게임 환경보다 향상된 게임 환경을 제공한다. 이는 게임 도메인에서 중요한 요소 중 하나라고 볼 수 있는 유저들의 체류 시간, 즉 유저 잔존율을 높일 수 있을 것이다. 유저의 게임 플레이 데이터가 담긴 패킷을 분석하여 게임과 유저에 대한 데이터를 확보하고 이를 활용하여 기존 게임보다 향상된 플레이 환경을 조성하여 유저 잔존율을 높인다면 이와 밀접하게 연결되어 있는 게임의 전반적인 수익 창출과 마케팅 측면에서 이점을 기대할 수 있을 것이다.

## 2.2기술 선택 및 사용한 오픈 소스 소개

- Wireshark: 네트워크 패킷 분석 도구로써, 해당 프로젝트에서는 Stardew Valley 에서 전송되는 패킷 데이터를 캡처하여 위치 데이터를 찾았고 해당 패킷의 구조, 정보들을 분석하였다.
- WinPcap: Windows 운영 체제에서 패킷 캡처를 위한 라이브러리 및 드라이버이며 Wireshark 와 함께 사용하여 Stardew Valley 의 패킷 데이터를 캡처하고 분석하였다
- SMAPI (Stardew Modding API): SMAPI 는 Stardew Valley 의 모드 개발을 위한 프레임워크이다. C# 언어로 작성된 SMAPI 는 게임의 기능을 확장하고 개인화할 수 있는 인터페이스를 제공한다. 이 프로젝트에서는 Stardew Valley 의 게임 내 데이터에 접근하고 수정하기 위해 SMAPI 를 사용하였다.
- Visual Studio : Microsoft 가 제공하는 통합 개발 환경(IDE)로 이 프로젝트에서는 Visual Studio 에서 C 로 패킷 데이터를 받아오고 분석한 데이터를 출력하며 실시간 위치 데이터가 저장된 파일을 생성하여 처리했다. 또한, Unity 를 통해 게임 객체의 위치 정보를 추출하고 미니맵에 표시하기 위해 C#의 다양한 기능과 라이브러리를 활용하여 미니맵을 동적으로 업데이트하고 사용자 인터페이스를 구성하는 등의 작업을 수행하였다.
- "https://github.com/WeDias/StardewValley"는 프로젝트에서 사용된 오픈 소스 저장소이다. 해당 저장소는 Stardew Valley 게임의 개선 및 확장을 위해 개발된 커뮤니티 기반의 리소스이다. 이 프로젝트에서는 해당 github 의 코드와 리소스를 활용하여 Stardew Valley 의 데이터 추출과 미니맵 기능 구현에 도움을 받았다.
- Lidgren.Network은 C#으로 작성된 오픈 소스 네트워킹 라이브러리이다. 주로 게임 개발에 사용되며, 게임 서버와 클라이언트 간의 네트워크 통신을 처리하는 데 특화되어 있다. 이 라이브러리는 UDP(User Datagram Protocol) 기반의 네트워크 통신을 지원하며, 데이터 패킷의 송수신, 연결 관리, 압축, 암호화 등 다양한 기능을 제공한다. 이 라이브러리가 StardewValley 멀티플레이 구현에 이용되어서

- Winsock2는 TCP/IP 프로토콜 스택과 통신하기 위한 함수와 구조체를 포함하고 있습니다. 이를 통해 애플리케이션은 인터넷이나 로컬 네트워크를 통해 데이터를 전송하고 수신할 수 있다. Winsock2는 네트워크 관련 작업을 쉽게 처리할 수 있는 추상화 계층을 제공하여 개발자가 네트워크 프로그래밍을 더 쉽게 구현할 수 있도록 도와준다.

  다음은 본 프로젝트를 완성하기 위해 사용한 해당 라이브러리의 특징들이다.

  1. 주소지정과 연결 설정: Winsock2를 사용하여 소켓에 IP 주소와 포트 번호를 할당하고, 서버에 연결하거나 클라이언트로부터의 연결 요청을 수락하는 기능을 구현할 수 있다.
  2. 데이터 전송과 수신: 소켓을 통해 데이터를 전송하고 수신하는 함수를 제공합니다. TCP/IP 프로토콜을 사용하는 연결 지향적인 통신과 UDP 프로토콜을 사용하는 비연결형 통신을 모두 지원한다.
  3. 비동기적인 통신: Winsock2는 비동기적인 소켓 통신을 구현할 수 있는 기능을 제공합니다. 이를 통해 네트워크 통신이 프로그램의 실행 흐름을 차단하지 않고 백그라운드에서 처리될 수 있다.

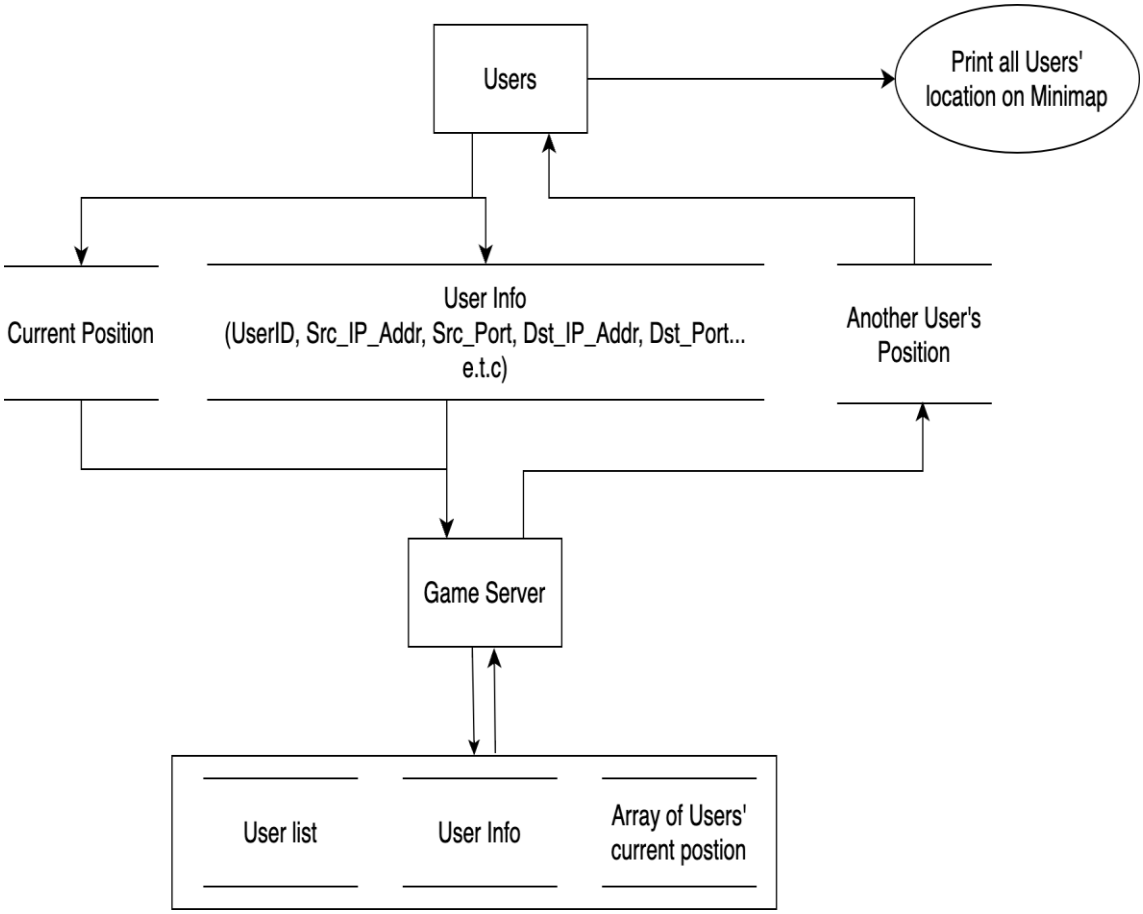
# 3. 프로젝트 설계

## 3.1 아키텍처 개요

이 프로젝트에서는 패킷 구조를 파악하기 위해 Reverse Engineering 방법론을 채택하였다. Reverse Engineering 방법론은 게임 클라이언트나 서버에서 전송되는 패킷의 구조와 데이터를 직접 분석하는 과정을 포함한다. 이를 위해 게임 클라이언트나 서버의 코드나 메모리를 역분석하여 패킷의 구조를 파악하고, 각 필드의 의미와 값의 범위 등을 분석하는 이러한 분석과정을 통해 패킷 데이터를 이해하고 추출할 수 있다. Reverse Engineering 방법론은 패킷 분석에 중요한 역할을 하며, 게임의 동작 원리를 이해하고 개선하는데 도움을 준다.

또한, Reverse Engineering 방법론을 이용하여 찾기 어려운 패킷은 Wireshark를 이용하여 패킷들을 상당량 캡쳐하여 Known Plaintext 방법론을 이용하여 분석하였다. Reverse Engineering 방법론을 통해 얻어낸 알려진 값을 이용하였다.

본 보고서에서는 시스템의 전체적인 구성을 파악할 수 있으며, 데이터의 흐름과 처리 과정을 시각적으로 보여주는 DFD 를 활용하여 프로젝트의 아키텍처와 구성 요소 간의 관계를 명확히 표현한다.




## 3.2 데이터 분석 및 처리 과정

1. 패킷 데이터 수집: Wireshark 와 WinPcap 을 사용하여 Stardew Valley 게임의 패킷 데이터를 캡처하고 저장했다.

2. 패킷 데이터 분석: 수집한 패킷 데이터를 분석하여 Reverse Engineering 방법론을 통해 게임 클라이언트나 서버의 코드를 역분석하여 패킷의 구조를 파악했다.

3. 필요한 데이터 추출: 분석한 패킷 데이터에서 필요한 정보인 좌표 데이터 x, y 를 식별하여 추출하였다.

- Player 가 좌측으로 이동합니다. (x 좌표 변동)

그림 1

그림 2

그림 3

그림 4

Microsoft Visual Studio 디버그 콘솔

```
============IP HEADER============
Src IP Addr : 192.168.123.106
Dst IP Addr : 192.168.123.152

============DATA============
43 6a 01 50 02 00 49 23         ba d1 a1 b7 9d 49 3d 00
00 00 49 23 ba d1 a1 b7         9d 49 02 00 15 00 00 00
00 00 00 00 27 00 00 00         8f 01 02 00 00 00 00 00
00 00 04 00 00 00 00 00         00 00 00 00 03 05 44 16
8e 45 7f 2e 00 44 01 f0         a8 2b 08 00 00 00 00 00
============LOCATION============
44 16 8e 45
7f 2e 00 44

============IP HEADER============
Src IP Addr : 192.168.123.106
Dst IP Addr : 192.168.123.152

============DATA============
43 6c 01 50 02 00 49 23         ba d1 a1 b7 9d 49 3d 00
00 00 49 23 ba d1 a1 b7         9d 49 02 0c 15 00 00 00
00 00 00 00 27 00 00 00         8f 01 02 00 00 00 00 00
00 00 04 00 00 00 00 00         00 00 00 00 03 05 d6 18
8d 45 7f 2e 00 44 00 b0         a9 2b 08 00 00 00 00 00
============LOCATION============
d6 18 8d 45
7f 2e 00 44

============IP HEADER============
Src IP Addr : 192.168.123.106
Dst IP Addr : 192.168.123.152

============DATA============
43 70 01 50 02 00 49 23         ba d1 a1 b7 9d 49 3d 00
00 00 49 23 ba d1 a1 b7         9d 49 02 7b 15 00 00 00
00 00 00 00 27 00 00 00         8f 01 02 00 00 00 00 00
00 00 04 00 00 00 00 00         00 00 00 00 03 05 5c c4
8c 45 7f 2e 00 44 01 a0         b0 2b 08 00 00 00 00 00
============LOCATION============
5c c4 8c 45
7f 2e 00 44

============IP HEADER============
Src IP Addr : 192.168.123.106
Dst IP Addr : 192.168.123.152

============DATA============
43 72 01 50 02 00 49 23         ba d1 a1 b7 9d 49 3d 00
00 00 49 23 ba d1 a1 b7         9d 49 02 87 15 00 00 00
00 00 00 00 27 00 00 00         8f 01 02 00 00 00 00 00
00 00 04 00 00 00 00 00         00 00 00 00 03 05 b1 9c
8b 45 7f 2e 00 44 00 60         b1 2b 08 00 00 00 00 00
============LOCATION============
b1 9c 8b 45
7f 2e 00 44
```

그림 5 `44 16 8e 45` `d6 18 8d 45` `5c c4 8c 45` `b1 9c 8b 45`

그림 6 `7f 2e 00 44`

그림 그림 1 → 그림 2 → 그림 3 순으로 진행했을 때 C 언어로 패킷 데이터를 캡쳐한 화면의 일부이다.

그림 5 를 보면 플레이어가 왼쪽으로 이동할 때, 특정 데이터가 지속적으로 감소하는 것을 관찰할 수 있으며, 이 데이터를 x 데이터로 추론할 수 있다. 그림 6 에 표시된 데이터는 변화가 없어 y 데이터임을 유추할 수 있다.

- Player 가 하단으로 이동합니다. (y 좌표 변동)

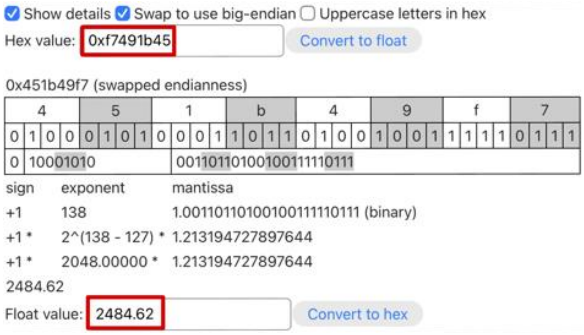
그림 1

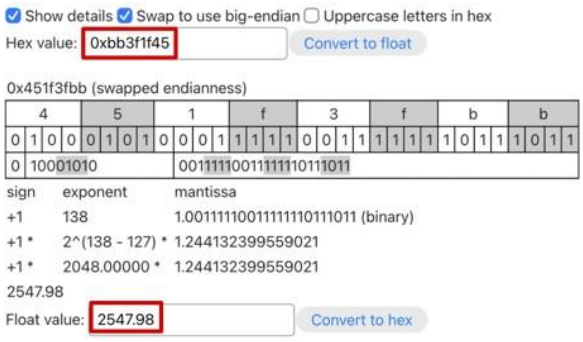

그림


그림 3


그림 4

```
Y : 5f 87 03 45
Y : 23 7d 07 45
Y : e7 72 0b 45
Y : ab 68 0f 45
Y : 6f 5e 13 45
Y : 33 54 17 45
Y : 33 54 17 45
Y : f7 49 1b 45
Y : bb 3f 1f 45
Y : bb 3f 1f 45
Y : 7f 35 23 45
Y : 43 2b 27 45
Y : 07 21 2b 45
Y : 07 21 2b 45
Y : cb 16 2f 45
Y : 8f 0c 33 45
Y : 8f 0c 33 45
Y : 8f 0c 33 45
Y : 53 02 37 45
Y : 17 f8 3a 45
Y : db ed 3e 45
```



x 좌표와 마찬가지로, 플레이어가 하단으로 이동할 때 증가하거나 감소하는 데이터를 관찰했다. 위의 그림 4 에 있는 해당 값이 y 좌표를 나타내는 데이터라고 추론했지만, 아래로 내려가는 플레이어와는 달리 y 좌표로 추론한 데이터는 증가하는 경향을 보였다. 후에 미니맵을 구현할 때의 편의를 위해, -1 을 곱한 후 특정 값을 더해 하단으로 내려갈수록 감소하는 양수값을 가지도록 조정하였다.

4. 데이터 변환 및 가공: 패킷에서 추출한 이동 데이터는 big-endian 으로 정렬된 4 바이트 부동 소수점(float) 형식의 데이터임을 확인할 수 있다. 유니티에서 미니맵에 좌표를 나타낼 때 편의를 위해 이러한 데이터에서 하위 2 바이트만 추출하여 필터링(filtering)하기로 했다.

**그림 1**

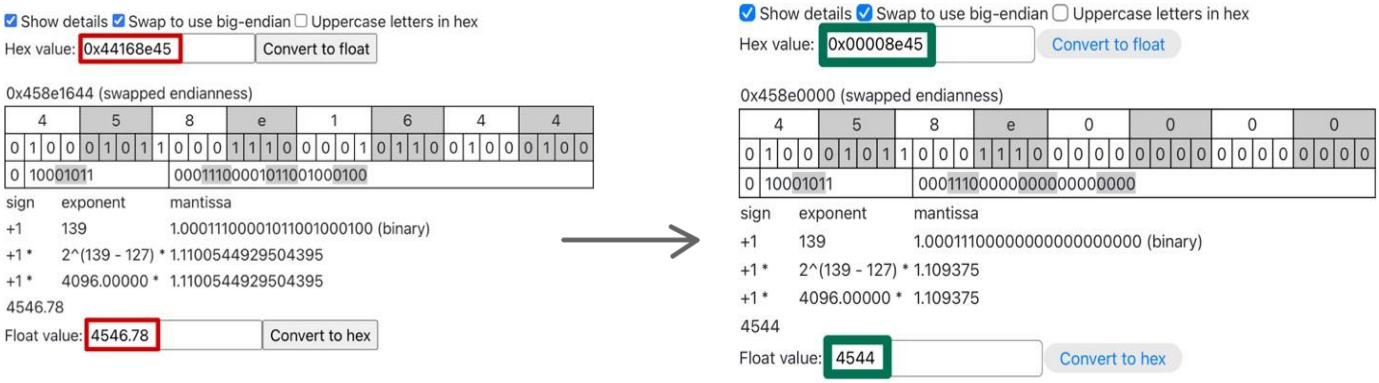



**그림 2**

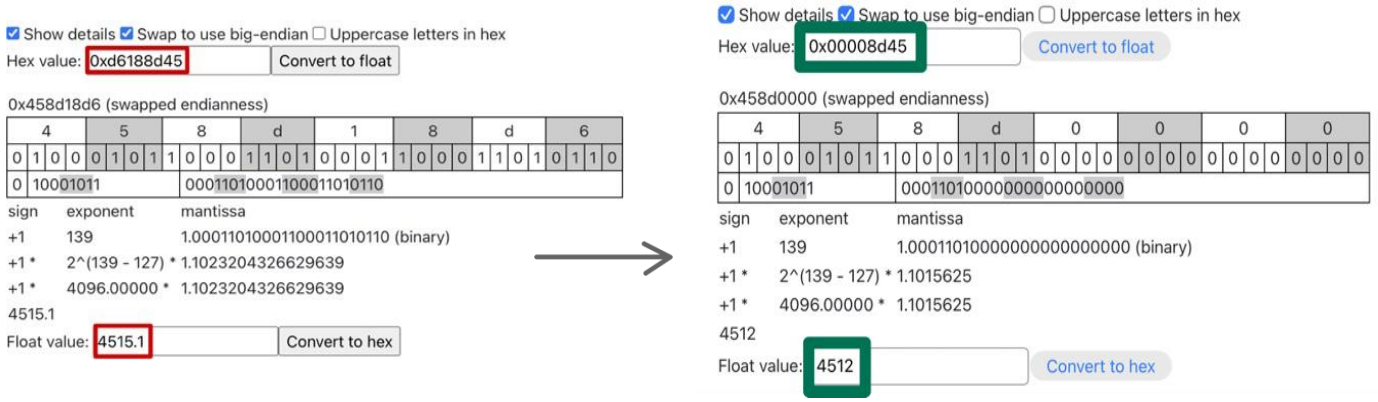


**그림 3**

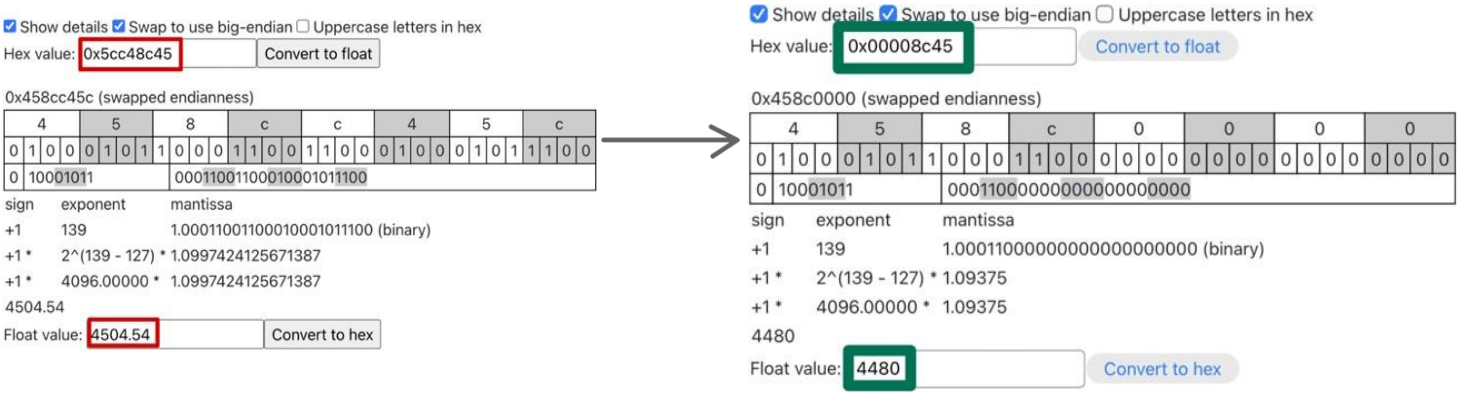
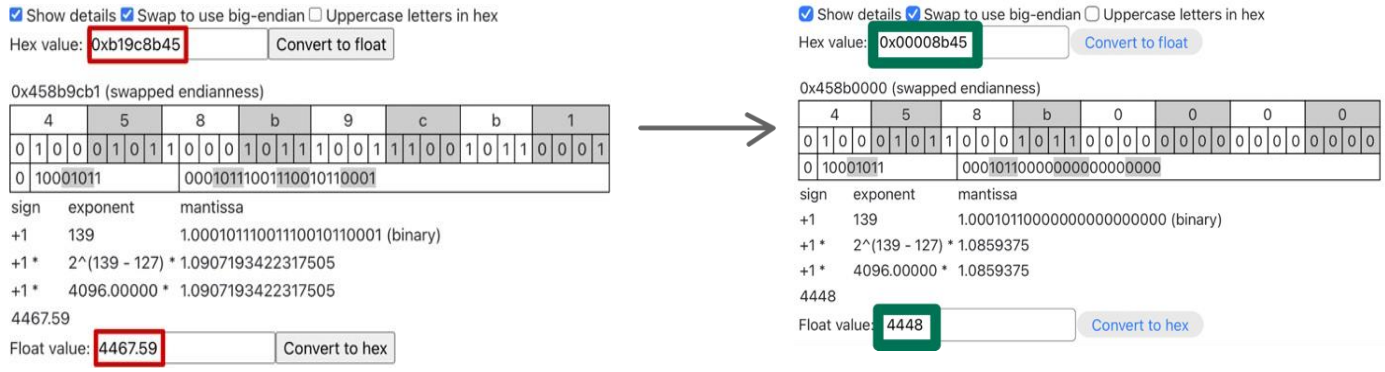



**그림 4**

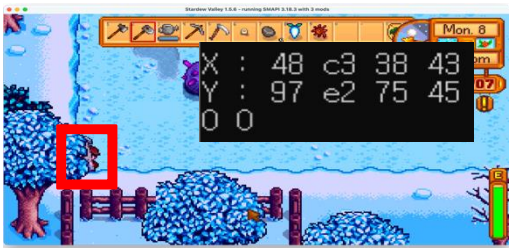



다음과 같이 4 바이트 데이터에서 하위 2 바이트만을 추출하여 정수(int)로 변환하였다

5. 데이터 처리 결과 확인: 변환 및 가공한 int 형 데이터를 확인하고 시각화 및 출력 등을 통해 데이터가 올바르게 처리되었는지 확인해 필요한 수정이나 보완 작업을 수행했다.


상단 좌측 데이터
X = 0, Y = 357


상단 우측 데이터
X = 610, Y = 309




최하층 좌측 데이터
X = 0, Y = 0


하층 우측 데이터
X = 608, Y = 24


상단 좌측 데이터
X = 184, Y = 576


상단 우측 데이터
X = 4608, Y = 512


최하층 좌측 데이터
X = 185, Y = 3904


하층 우측 데이터
X = 4864, Y = 3536

파란색 상자를 살펴보면, x 와 y 의 값을 int 로 변환했음에도 불구하고 '부적절한 값'으로 나타나는 것을 확인할 수 있다. 특히 y 는 배열에서의 y 와 달리 하단으로 갈수록 값이 작아졌다. 데이터를 더 쉽게 처리하기 위해, 최하층 좌측 데이터를 (0,0)을 기준으로 삼고 x 와 y 에 적절한 처리를 적용하여, 주황색 상자와 같은 결과를 얻었다.

또한, 모든 위치를 패킷으로 주고받는 것에는 한계가 있어 이동 데이터 간에는 빈 공간이 불가피하게 발생하는 것을 확인했다. 이러한 빈 공간으로 인해 미니맵 상에서 캐릭터의 이동이 끊겨 보일때가 있었고 이에 따라 더 부드러운 움직임을 구현하기 위해, 각 이동 방향마다 이동 데이터 간의 빈 공간을 채워주는 방법을 도입하여, 미니맵에서 캐릭터의 움직임이 자연스럽게 표현하였으나 오류가 많이 발생하는 등 제대로 작동하지 않아 이 문제는 해결하지 못해 제외했다.

이와같은 과정을 통해 데이터 분석 및 처리를 수행하여 게임의 필요한 정보를 추출하고 가공했고 이를 토대로 미니맵 구현 및 기능 개발에 활용했다.

### 3.3 미니맵 디자인 및 구성

미니맵 실행 시 시작화면 UI 에서 플레이어를 선택하여 시작할 수 있도록 <그림 1>과 같은 시작화면을 구성하였다. 타이틀 아래 플레이어를 선택할 수 있는 버튼을 만들어 미니맵 실행 시 카메라 포커싱 타겟을 정할 수 있게 구성하였다. 미니맵은 선택한 플레이어에 맞게 포커싱되어 해당 플레이어의 위치를 중점으로 보여지도록 설정하였으며 이 프로젝트에서는 3 명의 멀티플레이어 상황을 가정하여 미니맵을 설계하였다.

Player1 에 해당하는 플레이어는 흰색 닭, Playe2 에 해당하는 플레이어는 갈색 닭, Player3 에 해당하는 플레이어는 파란색 닭으로 설정하여 나타내었다. 예를 들어, Player1 선택 시 플레이어 1 의 위치를 중심으로 미니맵이 보여지고, 플레이어 2 선택 시 플레이어 2 의 위치를 중심으로 미니맵이 보여지도록 구현하였다. 각 플레이어에 따른 미니맵 화면은 아래의 그림과 같다.

- <그림 2>는 Player1 을 선택했을 시 Player1 에 해당하는 오브젝트인 흰색 닭의 위치를 중심으로 미니맵이 보여지는 모습이다.

- <그림 3>은 Player2 를 선택했을 시 Player2 에 해당하는 오브젝트인 갈색 닭으로 선택되어 미니맵이 Player2 의 위치를 중심으로 보여지는 모습이다.

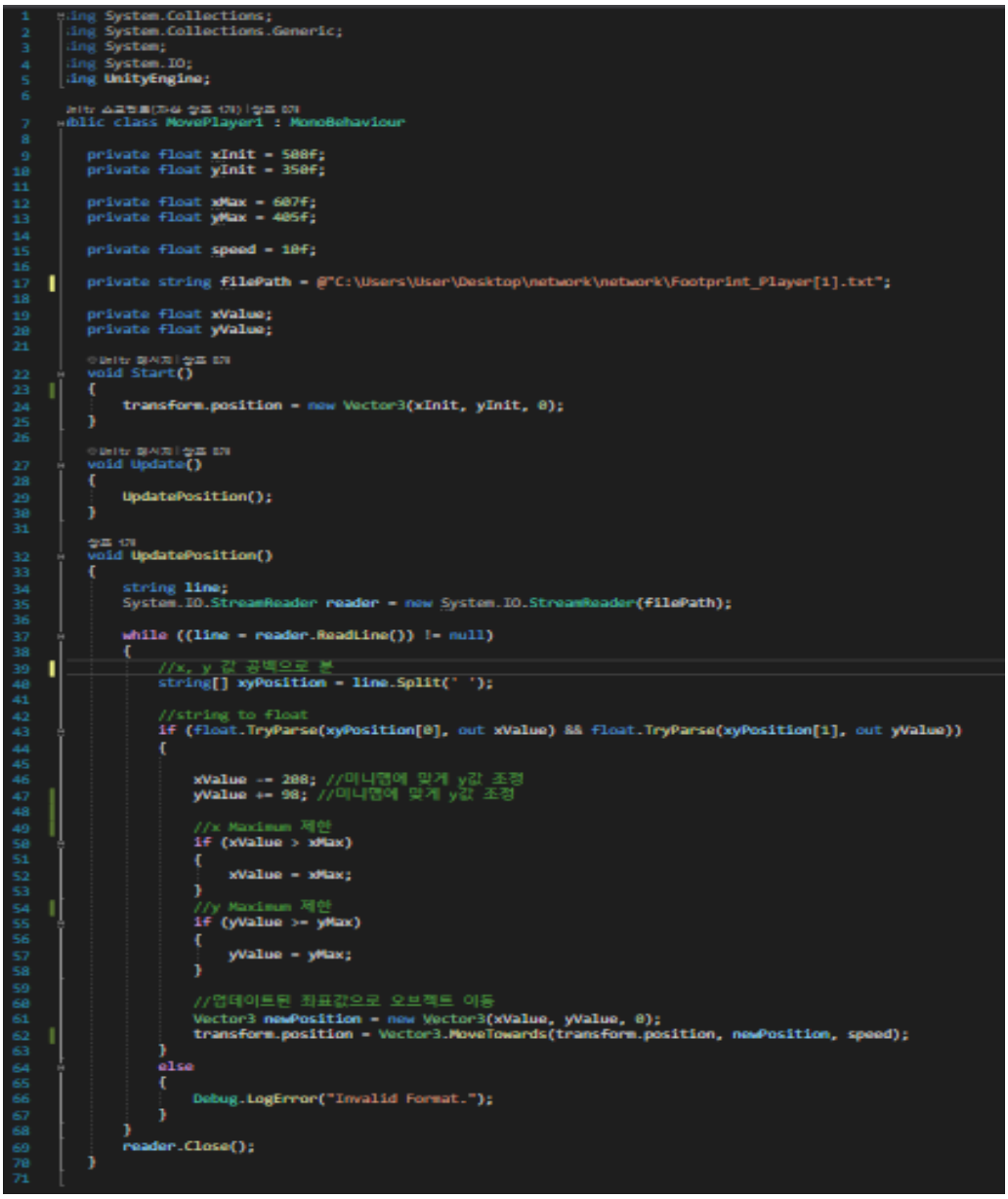

그림 1


그림 2


그림 3

### 3.4 미니맵 유니티 구현 및 코드 설명

- MovePlayer1.cs, MovePlayer2.cs, MovePlayer3.cs



```
using System.Collections;
using System.Collections.Generic;
using System;
using System.IO;
using UnityEngine;

public class MovePlayer1 : MonoBehaviour
{
    private float xInit = 508f;
    private float yInit = 350f;

    private float xMax = 607f;
    private float yMax = 405f;

    private float speed = 10f;

    private string filePath = @"C:\Users\User\Desktop\network\network\Footprint_Player[1].txt";

    private float xValue;
    private float yValue;

    void Start()
    {
        transform.position = new Vector3(xInit, yInit, 0);
    }

    void Update()
    {
        UpdatePosition();
    }

    void UpdatePosition()
    {
        string line;
        System.IO.StreamReader reader = new System.IO.StreamReader(filePath);

        while ((line = reader.ReadLine()) != null)
        {
            //x, y 값 공백으로 분
            string[] xyPosition = line.Split(' ');

            //string to float
            if (float.TryParse(xyPosition[0], out xValue) && float.TryParse(xyPosition[1], out yValue))
            {

                xValue -= 208; //미니맵에 맞게 y값 조정
                yValue += 98; //미니맵에 맞게 y값 조정

                //x Maximum 제한
                if (xValue > xMax)
                {
                    xValue = xMax;
                }
                //y Maximum 제한
                if (yValue >= yMax)
                {
                    yValue = yMax;
                }

                //업데이트된 좌표값으로 오브젝트 이동
                Vector3 newPosition = new Vector3(xValue, yValue, 0);
                transform.position = Vector3.MoveTowards(transform.position, newPosition, speed);
            }
            else
            {
                Debug.LogError("Invalid Format.");
            }
        }
        reader.Close();
    }

```

그림 4

각 플레이어 오브젝트(Player1, Player2, Player3)의 움직임을 제어하는 코드이다. <그림4>에 해당하는 코드는 Footprint_Player[1].txt 파일에서 Player1의 위치 정보를 읽어와 실시간으로 플레이어를 새로운 위치로 이동시키는 역할을 한다. Footprint_Player[1].txt 파일은 대괄호([ ]) 안에 있는 숫자에 해당하는 플레이어의 x, y 좌표 값이 공백을 기준으로 저장되어 있으며 이동 시 변하는 좌표 값으로 수정된다.

- Start() 함수는 스크립트가 시작될 때 플레이어 오브젝트의 초기 위치를 설정한다.

- Update() 함수는 매 프레임마다 UpdatePosition() 함수를 호출하여 플레이어의 위치를 지속적으로 업데이트 시켜준다.

- UpdatePosition() 함수는 filePath에 해당하는 경로에 있는 파일을 한줄씩 읽어오면서 공백(‘ ‘)을 기준으로 x와 y값으로 분리한 후 x, y값을 string에서 float형으로 변환하여 미니맵에 맞게 위치를 조정한다. 조정한 값이 최대값을 넘지 않도록 xMax, yMax로 제한하여 맵을 벗어나지 않게 하였으며, Vector3.MoveTowards() 함수를 사용하여 플레이어 오브젝트를 현재 위치에서 움직인 새로운 위치의 x, y 좌표로 일정한 속도로 이동시켜 플레이어의 위치를 업데이트 하도록 구현하였다.

- MoveCamera.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

Unity 스크립트(자산 참조 1개)|참조 0개
public class MoveCamera : MonoBehaviour
{
    public Transform target;

    public Button p1Button;
    public Button p2Button;
    public Button p3Button;

    public Transform Player1;
    public Transform Player2;
    public Transform Player3;

    public Camera MainCamera;

    private Text location;

    Unity 메시지|참조 0개
    private void Start()
    {
        // 버튼 클릭 이벤트 핸들러
        p1Button.onClick.AddListener(() => SetMainCameraTarget(Player1));
        p2Button.onClick.AddListener(() => SetMainCameraTarget(Player2));
        p3Button.onClick.AddListener(() => SetMainCameraTarget(Player3));

        MainCamera = GetComponent<Camera>();
    }

    참조 3개
    private void SetMainCameraTarget(Transform targetPlayer)
    {
        //플레이어에 맞게 카메라 포커싱
        MainCamera.transform.position = targetPlayer.position;
        target = targetPlayer;
    }

    Unity 메시지|참조 0개
    void LateUpdate()
    {
        transform.position = new Vector3(target.position.x, target.position.y, -10f);

    }
}
```

MoveCamera.cs는 플레이어에 따른 카메라의 이동을 제어하는 스크립트이다. 시작 화면 UI에서 3개의 버튼 중 플레이어에 맞게 클릭하면 해당 플레이어로 메인 카메라가 이동하고 LateUpdate() 함수에서 카메라의 위치가 플레이어를 따라가게끔 만들어 선택한 플레이어를 따라가는 카메라를 구현하였다.

- Start() 함수에서 p1Button, p2Button, p3Button 각각의 버튼이 클릭되면 해당 플레이어를 메인 카메라의 타겟으로 설정하는 이벤트를 연결한다. 그리고 GetComponent<Camera>()를 사용하여 스크립트가 연결된 게임 오브젝트의 Camera 컴포넌트를 가져와 MainCamera에 저장한다.

- SetMainCameraTarget() 함수는 선택한 플레이어에 따라 메인 카메라의 위치와 타겟을 설정한다. 선택한 플레이어의 위치로 메인 카메라를 이동시키고, target 변수를 선택한 플레이어의 Transform으로 설정한다.
- LateUpdate() 함수는 target.position을 기준으로 카메라가 플레이어의 위치를 따라가도록 설정한다.

# 구현

## 4.1 개발 환경 소개

- 게임 및 애플리케이션 개발을 위한 종합적인 개발 환경 요소들을 제공하는 유니티(Unity)에서 미니맵 제작을 진행하였다. 유니티는 다양한 플랫폼에서 실행될 수 있는 2D 및 3D 게임을 만들 수 있는 기능과 도구를 제공하며 PC, 모바일 기기, 콘솔 등 다양한 플랫폼을 지원하는 통합 개발 환경(IDE)이다. 유니티 개발 환경의 핵심 도구인 Unity Editor 는 게임 오브젝트를 시각적으로 조작하고 Scene 을 구성하며 스크립트를 작성하고 디버깅할 수 있는 기능을 제공한다. 또한 Unity Engine 은 기술적 세부 사항을 다루어준다. 유니티 환경에서는 게임 오브젝트의 동작을 제어하기 위해 C# 스크립트를 사용하며, 이를 통해 게임 오브젝트의 이동, 상호작용 등을 프로그래밍적으로 구현할 수 있다. 이 프로젝트에서는 위와 같은 유니티 환경을 기반으로 footPrint 미니맵 기능 및 동작의 구현을 위해 유니티에서 제공하는 Unity Editor, Unity Engine, C# 스크립트 등를 사용하였다.

## 4.2 구현 설명

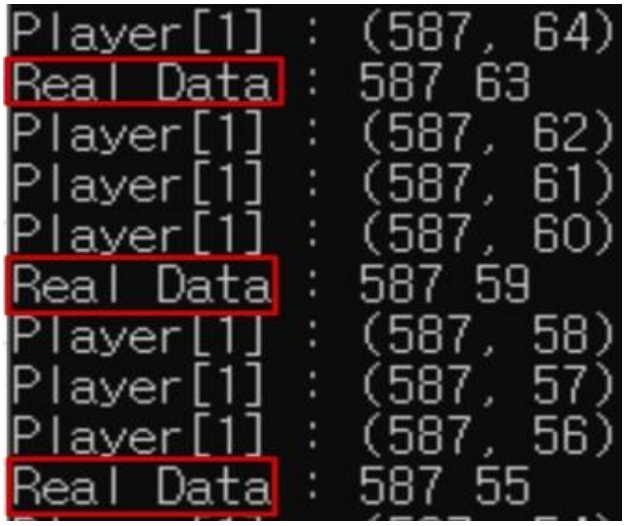
앞서 언급했던 이동 데이 간의 빈 공간을 채워주는 방법에 대해 설명하겠습니다. Player 는 총 8 개의 방향으로 움직일 수 있으며, 이는 사용자의 편의를 위해 12 시, 1 시, 3 시, 5 시, 6 시, 시, 9 시, 11 시로 표현했다. y 축 방향의 움직임인 6 시와 12 시 방향으로 움직일 때 생기는 빈 공간을 채우는 코드를 실행하기 위해서는 다음 조건을 만족해야 한다. x 값은 past_x(이전 x 값, "Footprint_Player[%d].txt"에 저장된 x 값)와 동일해야 하며, 동시에 y 값은 past_y(이전 y 값, "Footprint_Player[%d].txt"에 저장된 y 값)보다 작아야 한다. 이 조건을 충족하면 해당 Player 는 6 시 방향으로 움직이고 있는 것이다. 따라서 for 문을 사용하여 past_y-1 부터 현재 y 값 직전까지 printf 로 출력하고, fprintf 를 통해 각 줄마다 Footprint_Player[%d].txt 를 업데이트한다. 반대로, y 값이 past_y 보다 크다면 Player 는 12 시 방향으로 움직이고 있는 것을 의미한다 해당

Player 의 past_y+1 부터 y 직전까지 printf 와 fprintf 를 실행하면, 연속된 y 축 이동 데이터가 담긴 패킷을 받은 것과 같은 효과를 코드로 구현할 수 있다.

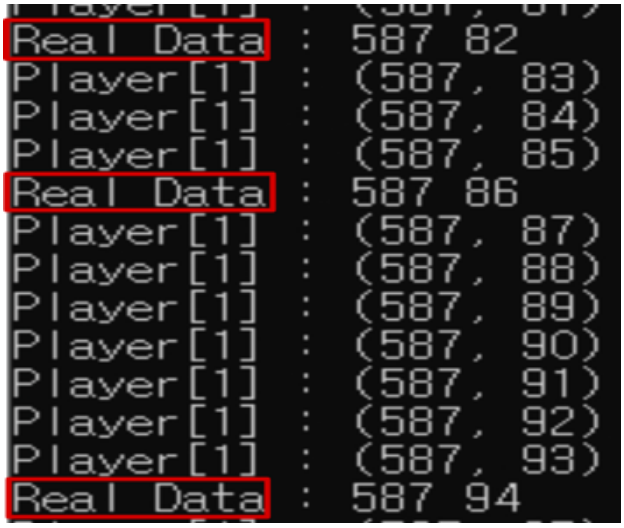
x 축 이동은 y 축 이동과 반대로 동작한다. 이동 데이터의 y 값은 각 플레이어의 이전 y 값과 동일해야 하며, x 값과 past_x 의 대소 관계에 따라 동작이 결정된다. 만약 x 값이 past_x 보다 작다면 플레이어는 9 시 방향으로 이동하고, x 값이 past_x 보다 크다면 3 시 방향으로 이동한다.

```
269                 if (x == past_x[player_num - 1] && y != past_y[player_num - 1]) { // y축 이동
270                     if (y < past_y[player_num - 1]) {                             // 6시 방향으로 이동중
271                         for (int new_y = past_y[player_num - 1] - 1; new_y > y; new_y--) {
272                             for (int i = 1; i < player_num; i++)
273                                 printf("\t\t\t\t");
274
275                             printf("Player[%d] : (%d, %d)\n", player_num, x, new_y);
276
277                             file = fopen(file_name, "w");
278                             fprintf(file, "%d %d\n", x, new_y);
279                             fclose(file);
280                         }
281                     } else { // 12시 방향으로 이동중
282                         for (int new_y = past_y[player_num - 1] + 1; new_y < y; new_y++) {
283                             for (int i = 1; i < player_num; i++)
284                                 printf("\t\t\t\t");
285
286                             printf("Player[%d] : (%d, %d)\n", player_num, x, new_y);
287
288                             file = fopen(file_name, "w");
289                             fprintf(file, "%d %d\n", x, new_y);
290                             fclose(file);
291                         }
292                     }
293                 } else if (x != past_x[player_num - 1] && y == past_y[player_num - 1]) { // x축 이동
294                     if (x < past_x[player_num - 1]) {                                    // 9시 방향으로 이동중
295                         for (int new_x = past_x[player_num - 1] - 1; new_x > x; new_x--) {
296                             for (int i = 1; i < player_num; i++)
297                                 printf("\t\t\t\t");
298
299                             printf("Player[%d] : (%d, %d)\n", player_num, new_x, y);
300
301                             file = fopen(file_name, "w");
302                             fprintf(file, "%d %d\n", new_x, y);
303                             fclose(file);
304                         }
305                     } else { // 3시 방향으로 이동중
306                         for (int new_x = past_x[player_num - 1] + 1; new_x < x; new_x++) {
307                             for (int i = 1; i < player_num; i++)
308                                 printf("\t\t\t\t");
309
310                             printf("Player[%d] : (%d, %d)\n", player_num, new_x, y);
311
312                             file = fopen(file_name, "w");
313                             fprintf(file, "%d %d\n", new_x, y);
314                             fclose(file);
315                         }
316                     }
317                 }
```

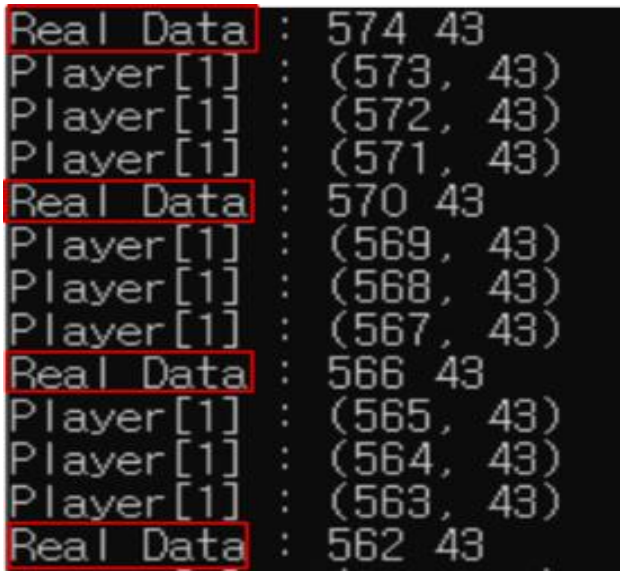
**6시 방향 데이터**



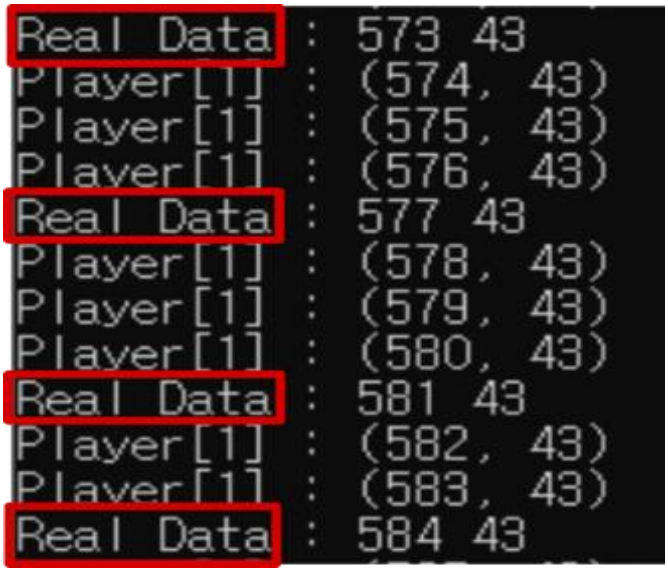

**12시 방향 데이터**




**9시 방향 데이터**




**3시 방향 데이터**




이와 같이 나눠지게 되며, 채워져야 할 이동 데이터의 x 값은 past_x[player_num-1]-1 또는 past_x[player_num-1]+1 부터 현재 x 값 직전까지의 범위이다. 이 범위에 대해 printf 와 fprintf 를 사용하여 출력하고 파일에 입력을 반복합니다.

X 축, Y 축 방향의 이동데이터가 아니면 대각선 방향의 움직임이라 생각할 수 있지만 x == past_x[player_num -1] && y == past_y[player_num-1]처럼 움직이지 않는 데이터가 숨겨져 있어서 꼭 x, y 둘다 이전 player 의 데이터와 같지 않다는 조건을 만족해야 대각선 방향의 이동을 했다고 볼 수 있다. 대각선 방향의 이동은 기울기가 1 또는 -1 에 가까워야 하므로, 1 시, 5 시, 7 시, 11 시

```
319                 else if (x != past_x[player_num - 1] && y != past_y[player_num - 1]) { // 대각선 이동
320
321                     if (x < past_x[player_num - 1]) {                                   // 왼쪽으로 이동중
322                         if (y < past_y[player_num - 1]) {                               // 7시 방향으로 이동중
323                             int new_y = past_y[player_num - 1] - 1;                     // 새로 찍힐 y값을 저장한 변수
324
325                             for (int new_x = past_x[player_num - 1] - 1; new_x > x; new_x--) {
326                                 for (int i = 1; i < player_num; i++)
327                                     printf("\t\t\t\t");
328
329                                 printf("Player[%d] : (%d, %d)\n", player_num, new_x, new_y);
330
331                                 file = fopen(file_name, "w");
332                                 fprintf(file, "%d %d\n", new_x, new_y);
333                                 fclose(file);
334
335                                 new_y--;
336                             }
337                         }
339                         else { // 11시 방향으로 이동중
340                             int new_y = past_y[player_num - 1] + 1;
341                             for (int new_x = past_x[player_num - 1] - 1; new_x > x; new_x--) {
342                                 for (int i = 1; i < player_num; i++)
343                                     printf("\t\t\t\t");
344
345                                 printf("Player[%d] : (%d, %d)\n", player_num, new_x, new_y);
346
347                                 file = fopen(file_name, "w");
348                                 fprintf(file, "%d %d\n", new_x, new_y);
349                                 fclose(file);
350
351                                 new_y++;
352                             }
353                         }
354                     }
```

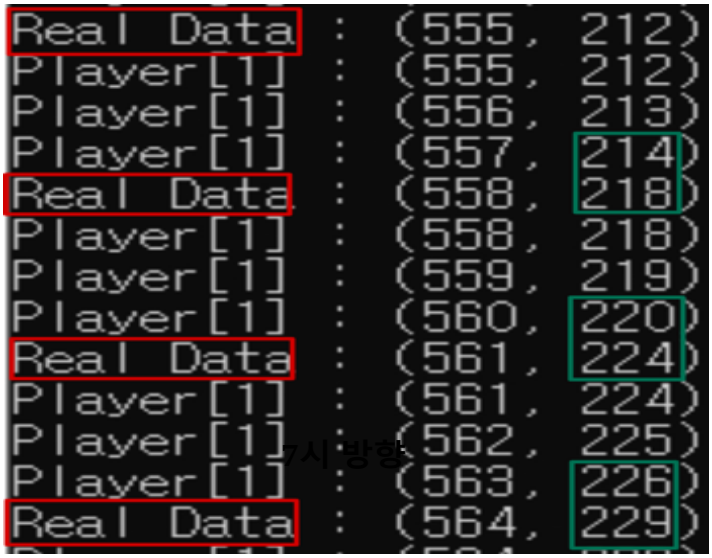
방향에 해당하는 데이터를 처리하기 위해 x 와 y 의 값은 실제 데이터를 입력 받으며, 기울기 1 또는 -1 을 가지는 값들을 x 와 y 직전까지 받아 처리한다. 그러나 float 을 int 로 형 변환하는 과정에서 버려진 데이터의 영향으로 인해 오류가 자주 발생하는데 이 점을 유의해야 한다.

또한, 1 시, 5 시, 7 시, 11 시 대각선 이동은 남아 있는데 원래의 4 바이트 float 형식의 이동 데이터가 강제로 int 로 형 변환되면서 값이 손실되므로 기울기가 완벽한 1 이 아니다. 그래도 기울기를 1 이라 가정하고 대각선 이동 시 발생하는 빈 데이터를 다음과 같은 방법으로 채워봤다.

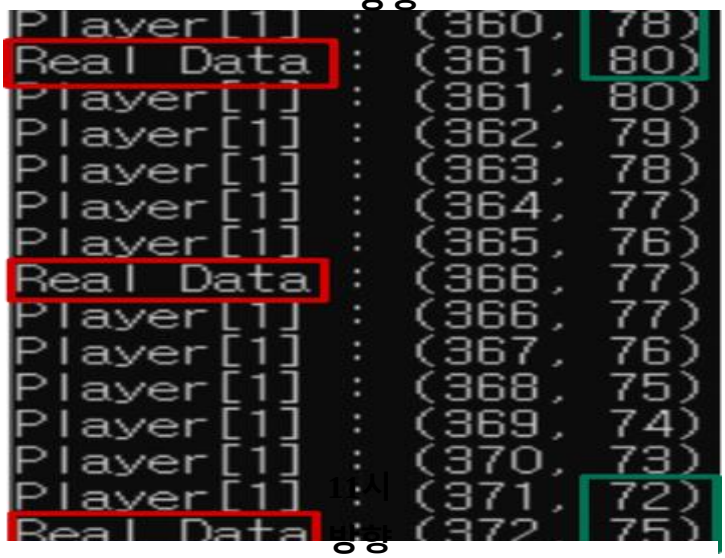
```
356                         else {                                      // 오른쪽으로 이동중
357                             if (y > past_y[player_num - 1]) {       // 1시 방향으로 이동중
358                                 int new_y = past_y[player_num - 1]; // 새로 찍힐 y값을 저장한 변수
359
360                                 for (int new_x = past_x[player_num - 1]; new_x < x; new_x++) {
361                                     for (int i = 1; i < player_num; i++)
362                                         printf("\t\t\t\t");
363
364                                     printf("Player[%d] : (%d, %d)\n", player_num, new_x, new_y);
365
366                                     file = fopen(file_name, "w");
367                                     fprintf(file, "%d %d\n", new_x, new_y);
368                                     fclose(file);
369
370                                     new_y++;
371                                 }
372
373                             } else {                                    // 5시 방향으로 이동중
374                                 int new_y = past_y[player_num - 1]; // 새로 찍힐 y값을 저장한 변수
375
376                                 for (int new_x = past_x[player_num - 1]; new_x < x; new_x++) {
377                                     for (int i = 1; i < player_num; i++)
378                                         printf("\t\t\t\t");
379
380                                     printf("Player[%d] : (%d, %d)\n", player_num, new_x, new_y);
381
382                                     file = fopen(file_name, "w");
383                                     fprintf(file, "%d %d\n", new_x, new_y);
384                                     fclose(file);
385
386                                     new_y--;
387                                 }
388                             }
```

하지만 사진의 초록색 박스처럼 예측하지 못한 또 다른 오류들이 계속해서 나와 제외했다.

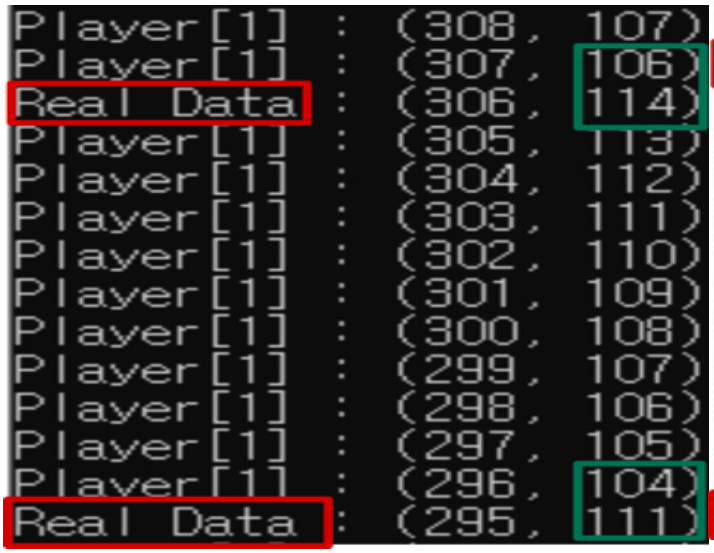
1시 방향



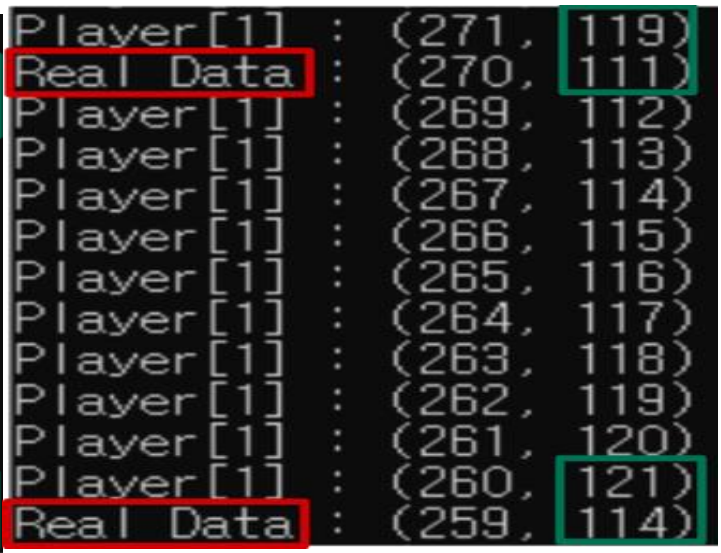

5시 방향




7시 방향




11시 방향



1. 구조체

-Ip_header: Internet Protocol Header의 내용을 저장하는 구조체

-udp_header: User Datagram Protocol Header의 내용을 저장하는 구조체

-data_header: 패킷의 data영역의 내용이 어떤 데이터인지 알려주는 헤더의 내용을 저장

```
unsigned char data_lidgren_type;//1
unsigned short data_squence;//2
unsigned short data_type;//2
```

Reverse Engineering 방법론을 사용한 결과, Stardew Valley는 Lidgren.Network API를 사용한다는 것을 알 수 있었다.

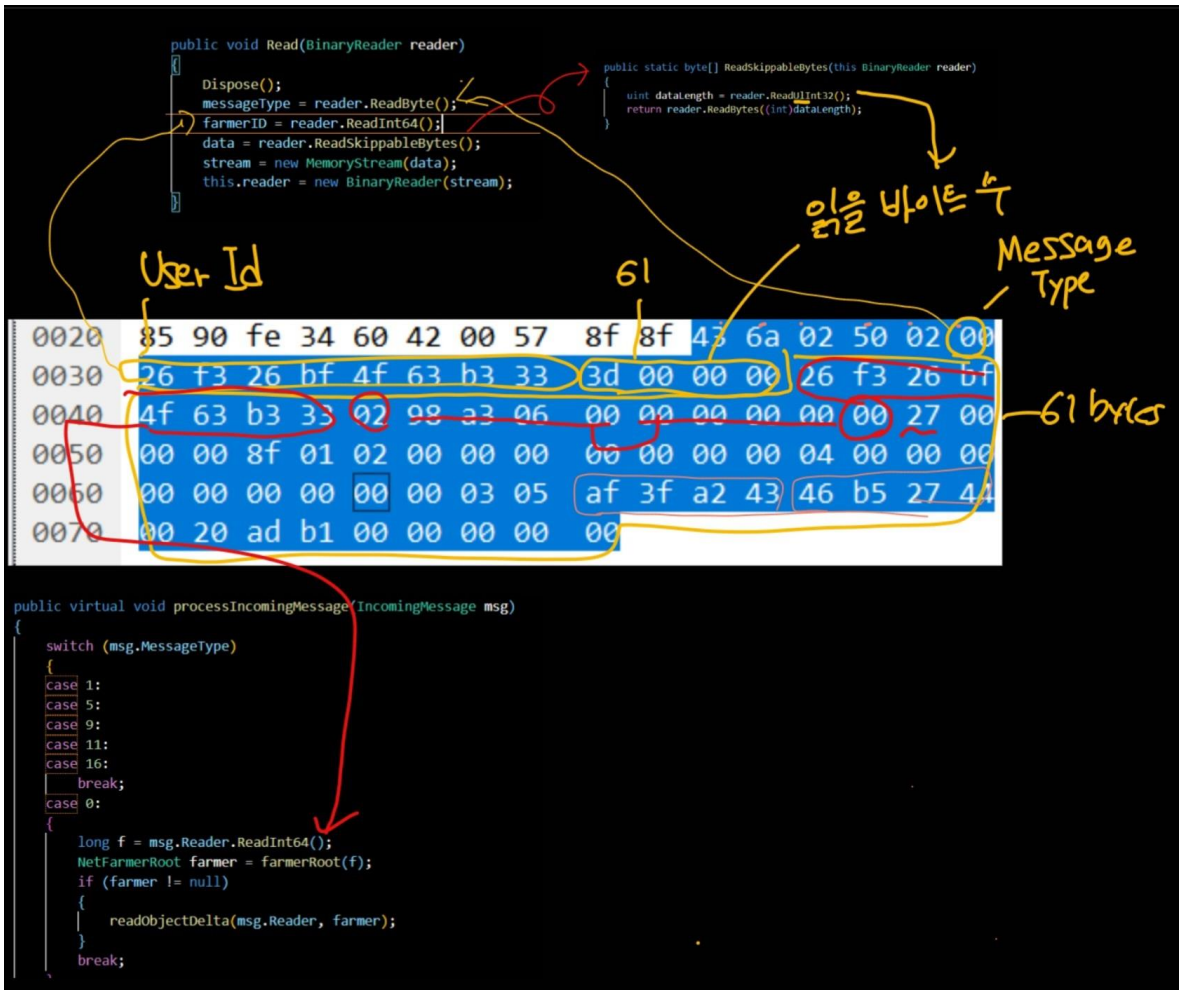
```
unsigned char data_messagetype;//1
unsigned long long data_userid; //8
unsigned int data_len;//4
```

Stardewvalley 의 자체코드의 Read()에서 알아낸 헤더의 순서와 크기이다.

```
unsigned int header_len;
```

위 구조체는 멤버 변수 중, 8 바이트가 넘는 것이 있어

컴파일러의 패딩 규칙을 수정하였다.



2. 함수

패킷을 분석하는 함수들은 두 가지 주요 목적을 가지고 구현되었다.

1. 좌표 데이터인가?

2. 여러 개의 패킷이 붙어오는 경우인가?

→여러 개의 패킷이 붙어오는 경우가 다수 있었다. 그 중에서 위치 데이터를 포함하는 패킷 또한 존재하였다. 이런 데이터를 무시한다면 데이터들 간의 간격이 커지기 때문에, 이를 구분하여 다른 패킷과 연결되어 있는 좌표 값을 추출하였다.

```
struct ip_header* check_ip_header(const unsigned char* data);
```

IP영역 헤더를 추출하고 구조체를 반환->길이를 알아낼 수 있음.

```
struct udp_header* check_udp_header(const unsigned char* data);
```

UDP영역 헤더를 추출하고, 포트번호 알아냄. UDP헤더 길이는 8 고정

```
struct data_header* check_data_header(const unsigned char* data, int len);
```

데이터 영역의 헤더를 확인하고 경우의 수에 따라 데이터 가공.

데이터 영역의 첫 번째 데이터는 lidgren.network API에서 제공하는 전송되는 타입을 구분할 수 있는 값이다.

첫 번째 바이트는 lidgren.network에서 결정된 패킷의 종류를 나타내는 숫자이다.

case 0x43(67):

userreliableordered 데이터를 의미한다.

뒤에 오는 2바이트 크기의 값을 통해 데이터들이 구분된다.

→0x2548일 때, 각 플레이어들에 대한 전체적인 정보가 전송된다.

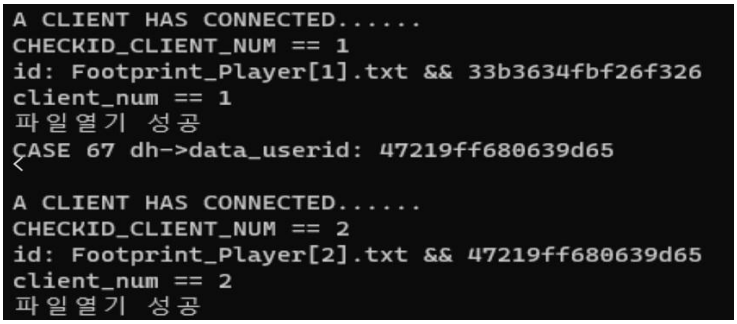
패킷의 최대 길이는 정해져 있기 때문에, 0x09값 다음에 가진 순서 값을 가지고 여러 번에 나눠서 전송된다. 이때, 만약에 순서 값이 0이라면, 2byte뒤부터 userid가 전달된다.

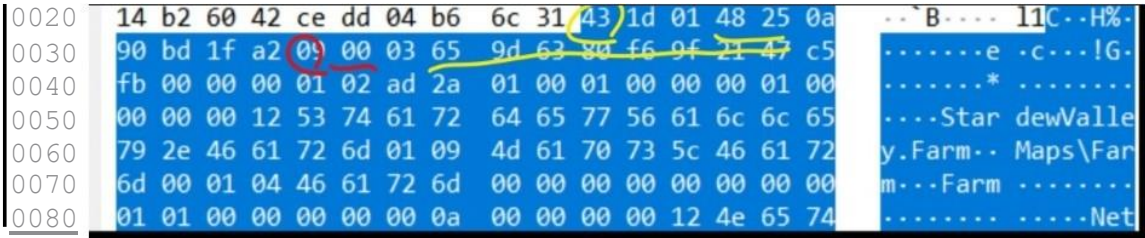
이러한 종류의 패킷은 user가 접속할 때 무조건 전송되므로, 이 패킷이 오면 파일이 생성되고, 터미널에 연결되었다고 표시하였다.

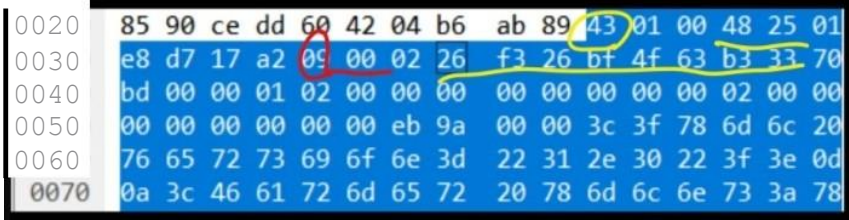
이는 Reverse Engineering을 시도하였다가 찾을 수 없어서, 여러 패킷들의 고정된 값과 변하지 않는 값, 알려진 값을 확인하여 해당 내용만 특정할 수 있는 고정 값을 찾아내는 방식으로 분석하였다.

→0x0068일 때, 서버에 접속한 유저가 3명 이상일 때, 한 유저가 접속을 종료하면 전체적으로 전송되는 type이다. 이를 이용하여 disconnect를 하는 기능을 구현하였다.

→0x01f8일 때, 헤더의 크기는 맞지만, 뒤에 전송되는 내용이 본 프로젝트와 무관하여 무시하는 패킷





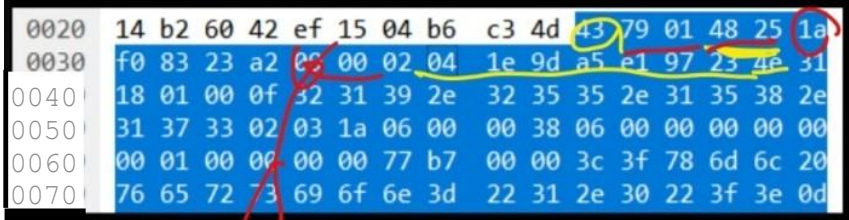

접속할때 서버가 모든 User정보
전 Client로 저장된 것 Server로
전송

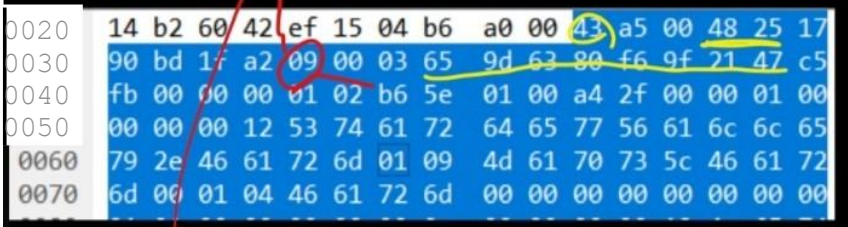
↳ Type가 0x2548
&&
data[0] = 0x09
data[1] = 0x00
= 데이터묶음
의미

data[13] 부터 8byte
= User id

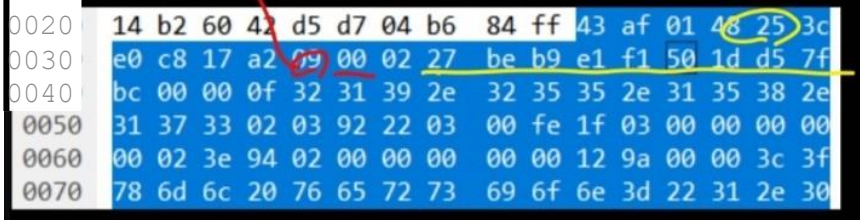
Or

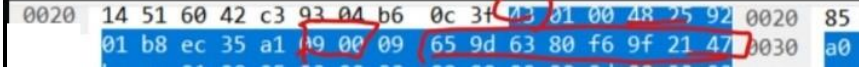
data[10] 부터 1byte
길이

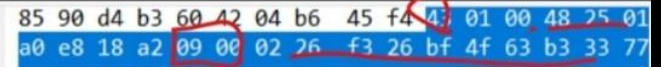






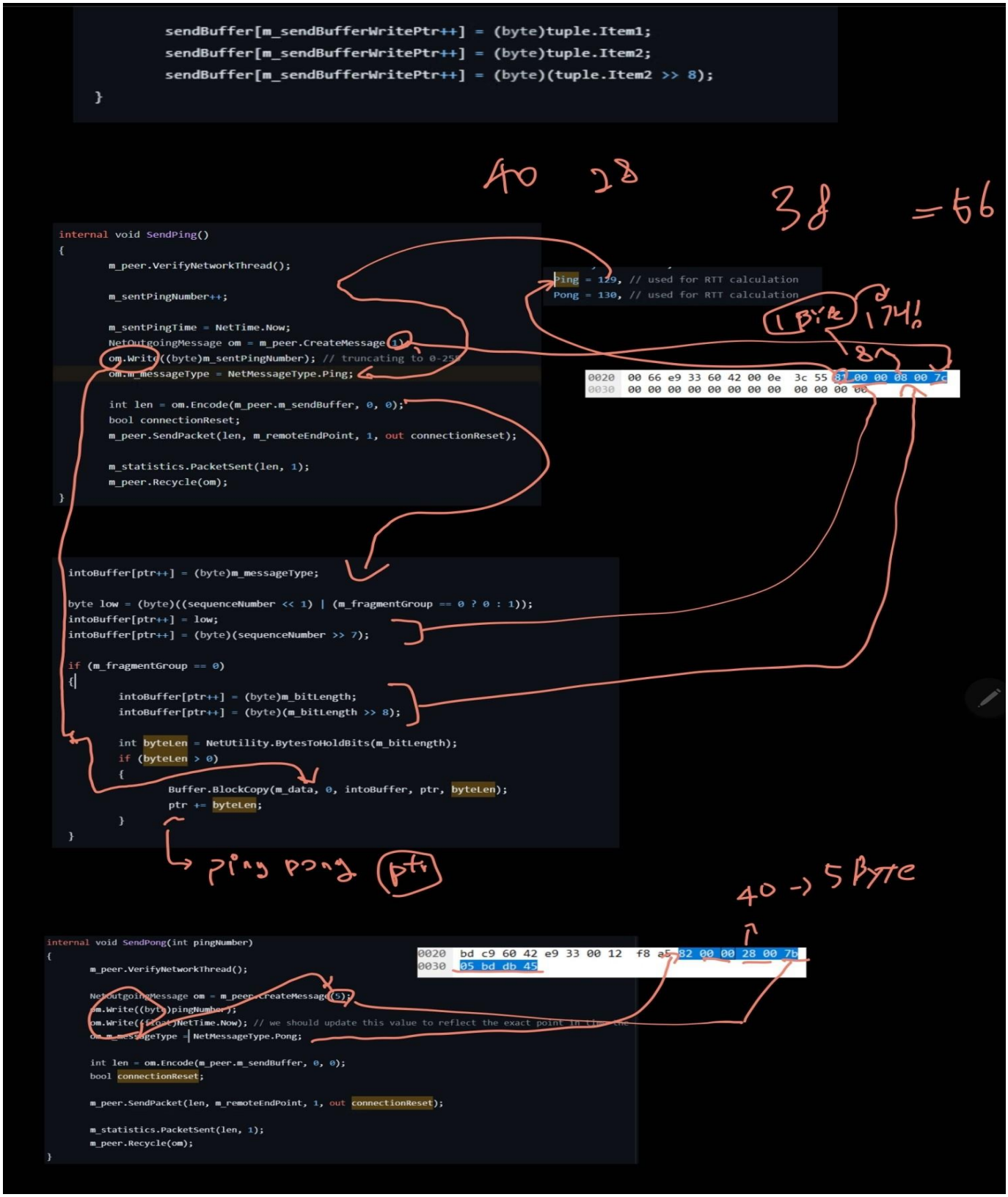



3. case 0x81:

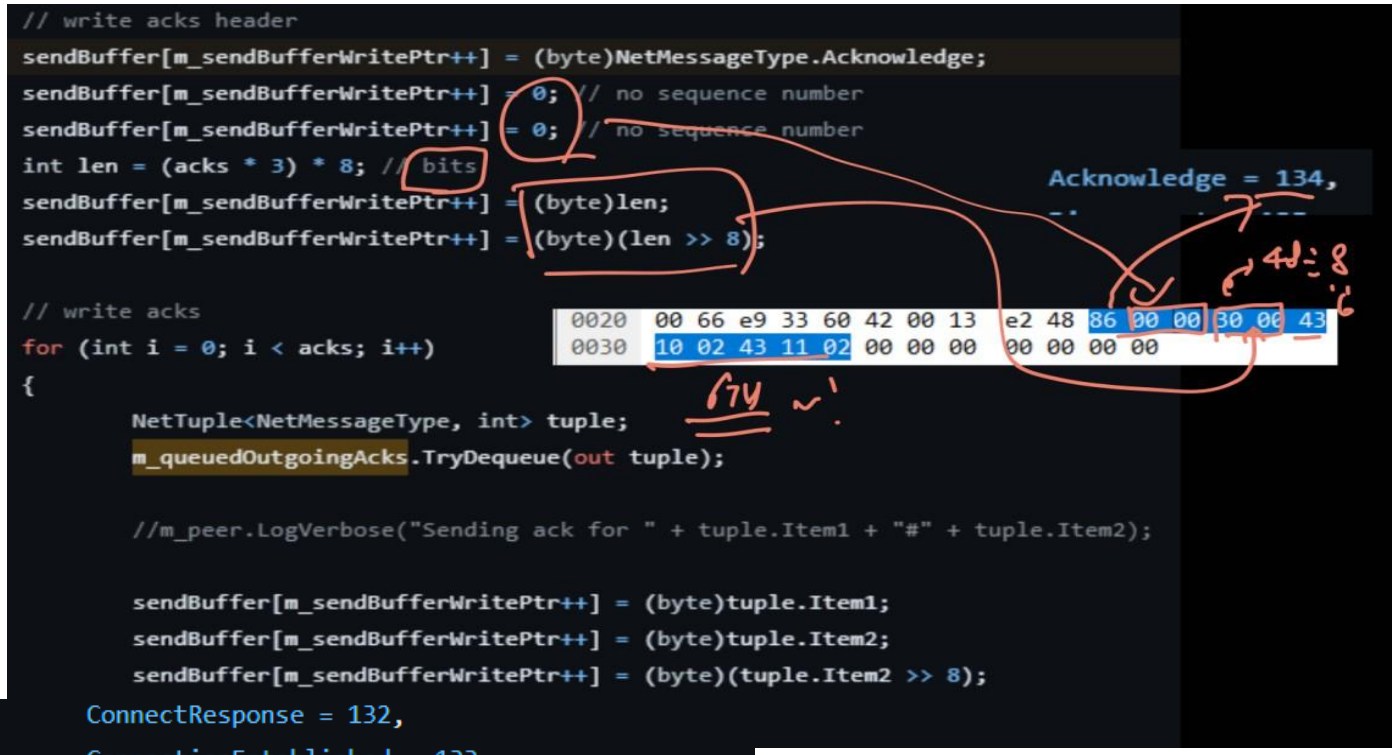
   case 0x82:

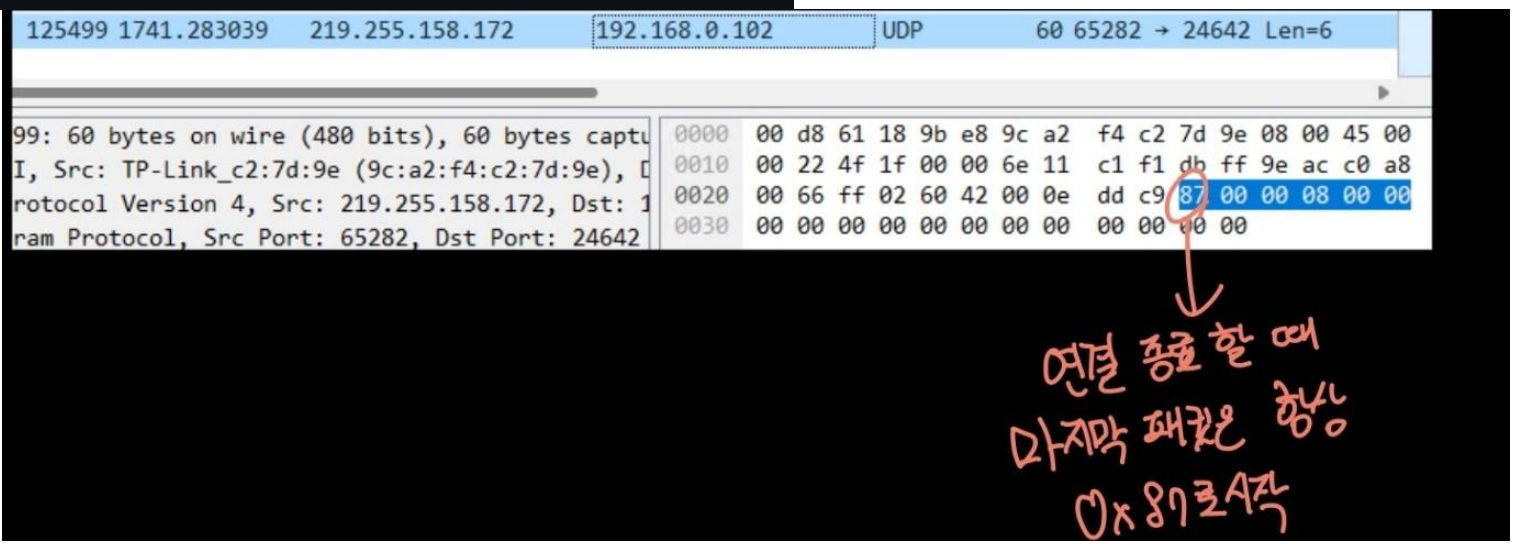
   Ping과 Pong이다. 서버와의 latency를 확인하기 위해 전송되는 패킷이다



4. case 0x86:
   ACK기능이다.

위 3개의 경우, 패킷의 전체 길이가 60으로 패딩되는 경우가 있어 처리해주었다.
case 0x83~89는 길이가 4번째, 5번째 바이트의 2배와 같아서 따로 처리해주었다.
0x87은 접속을 종료할 때 전송되는 패킷의 type이다.



void set_offset(const unsigned char** pkt_data, int num);

각 헤더를 확인한 후, 헤더의 길이만큼 데이터를 가리키는 포인터를 이동시켜준다.

int check_id(struct data_header* dh, long long* user_id_arr, int* current_usr);

이 함수에서는 전송된 데이터에 해당하는 플레이어가 누구인지 확인한다.

check_data_header()에서 설명한 것처럼, data_type 이 0x2548 일 때, 연결하는 경우이다.

0x47219ff680639d65 는 서버를 여는 계정의 id 이므로, 사용하지 않기로 결정했다.

연결된 user_id 는 배열에 저장되는데, 현재 전달받은 패킷의 userid 가 배열에 존재한다면, 이미 연결된 클라이언트이므로, 등록하는 과정을 생략한다.

0x87 은 disconnect 이고, 등록되지 않은 user 가 연결해제 하는 경우를 처리해주었다.

그리고 userid 가 유효할 경우에는 userid 배열에서의 현재 user 의 index 를 저장한다.

그리고 각 경우마다 main()에서 처리할 수 있도록 고유한 값을 return 한다.

void print_data(const unsigned char* data, int current_usr);

main()

패킷 캡쳐 후, 헤더를 확인한 뒤, 하나의 패킷이 처음부터 끝까지 다 분석된 후 다음 패킷을 받아 처리한다.

-1: 패킷이 유효하지 않거나, 필요 없는 패킷 일 경우.

1: 접속하는 user 의 파일 생성

2: 접속 종료 userid 출력.

3: 유효한 데이터. 위치 데이터 type 이라면 파일로 데이터를 출력.

4, 5: 필요 없는 데이터지만 userid 를 확인 가능할 경우. 붙어오는 패킷 확인을 위해 구분해 두었다.

함수 **Print_data** 는 다음과 같은 기능을 수행한다.

1. 이동 데이터가 들어간 파일 이름 형식인 "Footprint_Player[%d].txt"을 **temp1** 에 저장한다.
2. 하위 2 바이트를 빅 엔디안으로 계산하여 **x** 와 **y** 에 정수 형태로 저장한다.
3. 집안의 **y** 데이터는 334 에서 501 사이이고, 현관의 **y** 데이터는 250 이다. 이 두 값의 차이를 이용하여 플레이어가 집 현관(밖)에 있는지 여부를 판단한다. 만약 밖에 있다면,

각 플레이어의 **is_out** 변수를 **true** 로 설정하고, **past_x** 와 **past_y** 에 현재 **x** 와 **y** 값을 저장한다.

4. 플레이어가 다시 집으로 들어왔다면 **is_out** 변수를 **false** 로 변경한다.

만약 현관 데이터가 아니라면, 현재 위치 **x** 와 **y** 를 출력하고, "w"로 열린 각 플레이어 파일에도 **x** 와 **y** 를 한 줄로 출력합니다.

1. pcap_findalldevs_ex 함수를 사용하여 시스템에 연결된 네트워크 디바이스(네트워크 인터페이스) 목록을 검색한 후, 선택된 인터페이스를 사용하여 패킷 캡처를 수행하기 위해 해당 네트워크 디바이스를 연다.

2. Pcap_compile 함수로 패킷 필터 규칙을 컴파일 하는데 이때 FILTER_RULE 은 stardew vally 가 사용하는 패킷 프로토콜인 udp 로 하였다.

3. 그 후 선택된 디바이스를 pcap_open_live 로 열고 그것을 제어하기 위한 Handle 을 받았으므로 더 이상 그 디바이스에 대한 정보가 필요없으니 pcap_findalldevs 를 통해 생성된 Linked List 를 삭제한다.

4. 이후 while 문에서 pcap_next_ex 을 통해 네트워크 디바이스로부터 패킷을 읽어 'pkt_data'에 저장하고 'header'에 패킷의 메타데이터를 저장한다. Pcap_next_ex 은 패킷을 읽어오면 양수, buffer 가 비어있으면 0 을 리턴하는 pcap_next_ex 은 오류(리턴이 음수)가 나타날때까지 다음 패킷을 지속해서 가져온다.

5. pcap_next_ex 함수는 네트워크 디바이스로부터 패킷을 가져올 때, 패킷 헤더와 페이로드를 함께 반환하는데 offset 을 이용하여 패킷 데이터에서 페이로드의 특정위치로 갈 수 있다.

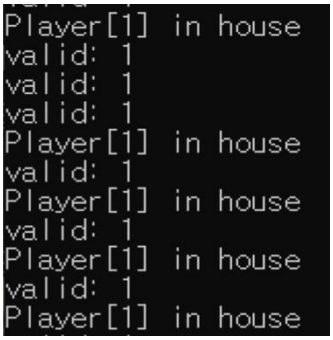
6. pkt_data 포인터를 offset 만큼 이동시키는 set_offset 함수를 통해 pkt_data 의 14 뒤에서 ip 출력하고 ip p_header 의 구조체 형태로 변환하고 ip 영역 만큼 pkt_data 를 또 옮긴다.

7. check_udp_header 로 udp 헤더를 확인하고 구조체 형태로 바꾼다.
해당 함수는 UDP 헤더 구조체를 확인하고, 포트 번호를 바이트 순서를 빅 엔디안으로 변경하여 수정한다.

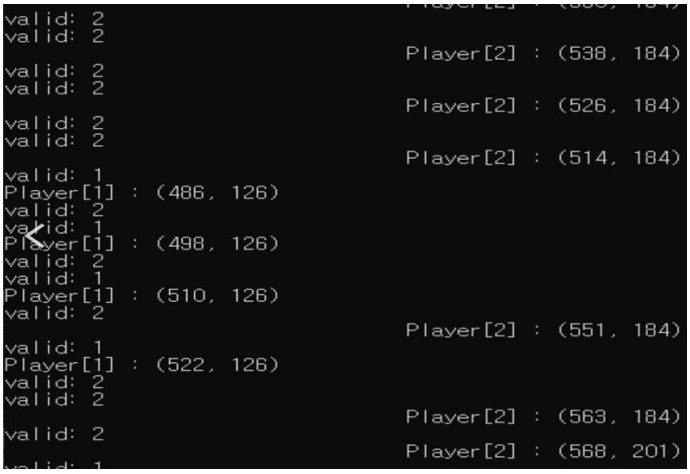
8. pkt_data 를 8 byte (udp 헤더) 이후의 데이터 영역으로 보낸다.

9. 만약 포트번호가 24642(서버이면)

- num = 14(패킷 헤더) + ih->ip_header_len * 4(IP 영역) + 8(udp 헤더)
- 데이터 헤더를 구조체 포인터 dh
- case 1 : connect ➔정보들 출력, 파일 포인터를 열어 현재 플레이어의 id 입력, client_num++;
- case 2 : disconnect ➔ 정보들 출력, 파일 포인터 close, client_num—
- case 3 : 정보있으면 print
- case 4, 5 : 헤더정보 있으나 필요없을때, 유효하지 않을때 break
- 나머지 덜 처리한 부분이 있으면 print 한다.




집을 오갈 때 **y** 위치 데이터의 큰 변화에 초점을 맞추고 있으며, 플레이어가 게임에 처음 접속할 때는 침대에서 시작하기 때문에 **past_x** 와 **past_y** 에 침대의 위치 데이터를 저장한다. 집에 돌아올 때는 **x** 의 위치 데이터의 큰 변화에 초점을 두어 구별했다.




- for (int i = 1; i < current_usr; i++) // 위 사진과 같이 player 별로 구별하기 쉽게 출력하기전에 띄어쓰기 함

pritf("₩t₩t₩t₩t");

## 4.3 실행결과 및 성능평가

- 게임 서버에 접속한 플레이어들의 위치 정보를 와이어샤크로 캡쳐한 패킷을 필터링 및 분석하여 추출하고 미니맵에 맞게 x, y 좌표 값을 조정하여 나타내었다. 게임 서버와 클라이언트 사이 오고가는 패킷을 분석하여 플레이어의 위치를 나타내는 데이터를 추출 및 가공하여 미니맵에 맞게 맵핑하여 자신의 움직임 뿐만 아니라 다른 플레이어의 이동 정보를 패킷에서 추출해 나타내는데 성공하였다.

- 다수의 플레이어들과 함께 상호작용하며 플레이하는 경우, 다른 플레이어의 이동 정보 또한 패킷에서 추출해 미니맵 상에 나타내었고 미니맵에서의 표시된 플레이어들의 위치 및 이동 시 변화한 좌표 값에 맞게 미니맵에 표현됨을 확인하였다. 또한, 게임 상에서의 위치와 패킷에서 추출한 x, y 좌표 값을 기반으로 나타낸 미니맵 상에서의 위치가 매우 흡사하게 나타나 플레이어들의 위치를 미니맵으로 확인 가능하였다.

## 4.4 한계점

1. 유니티에서 값을 읽어올 때, 한 번에 하나의 client 만 데이터가 적용되어 여러 플레이어가 이용할 때, 딜레이가 좀 있다. 여러 플레이어의 데이터를 동기화하는 작업이 필요하다.
2. Wincap 에서 한 번에 다량의 패킷이 오면 중간중간에 인식을 못하는 경우가 생김. 패킷 인식 속도의 한계가 있다.
3. Stardewvalley 는 소스코드가 공개되지 않아서 decompile 한 code 를 분석했어야 했다. 정확하지 않은 코드도 몇몇 있었다.
4. Reverse Engineering 과 고정/가변/고유 값을 통해 패킷의 특정성을 발현하는 데에는 너무 시간이 오래 걸린다