תיאור הפרוטוקול

Overview

בגדול כל פקטה שנשלחת בין בלקוח לבין השרת מחולקת ל3 חלקים.
חלק ראשון DataLength חלק שני Packet ID חלק שלישי Data

DataLength - חלק זה הוא int שמסמל את אורך המידע שבפקטה

PacketID - מספר הפקטה הוא int ולכן תופס את 4 הבייטים הראשונים.
מספר הפקטה מסמל את דרך הטיפול בפקטה הנתונה.

Data - מידע הפקטה מכיל את לא תאמינו... מידע הפקטה!

בפועל פקטה תיראה כך:

כאשר הבייטים הירוקים מייצגים את המידע שעובר בפקטה, הכחולים את מספר הפקטה והצהובים את אורך המידע.
*הערה: כאשר אנחנו כותבים מחרוזת אנחנו מוסיפים את אורכה בהתחלה (הדבר אינו משפיע על בחירת דרך הטיפול)

ועכשיו הצד המקבל קרא את 4 הבייטים השניים
מהפקטה ויודע שהוא צריך לטפל בה בדרך
מספר 1

עכשיו בואו נבין מה זה אומר טיפול בדרך "1" אבל קודם. איך מכינים פקטה לדרך "1"

אופן כתיבת וקריאת הפקטות

כתיבת הפקטות

בקוד תכונות המחלקה Packet נראות כך:
(קיימת תכונה נוספת שאינה חשובה להסבר הזה)

```
public class Packet : IDisposable
{
    private List<byte> buffer;
    private byte[] readableBuffer;
    private int readPos;
```

למחלקה קיימות 3 פעולות בונות.
כרגע נדבר על הפעולה הבונה השניה שנראת כך

```
/// <summary>Creates a new packet with a given ID. Used for sending.
10 references
public Packet(int _id)
{
    buffer = new List<byte>(); // Initialize buffer
    readPos = 0; // Set readPos to 0
    Write(_id); // Write packet id to the buffer
}
```

אנחנו נתמקד כרגע בפעולה הזו בגלל שהיא הפעולה שמשתמשים בה כדי להכין פקטה לשליחה.

הפעולה תיצור לנו פקטה עם ID
על פקטה זו נוכל לכתוב מידע בעזרת הפעולה ()Write של המחלקה.

דוגמה מהפרויקט ליצירת פקטה:

```
/// <summary>
/// sends a reqest for the server to kill a player
/// </summary>
/// <param name="_targetId"> player we want to kill ID</param>
1 reference
public static void AttemptKill(int _targetId)
{
    using (Packet _packet = new Packet((int)ClientPackets.attemptKill))
    {
        _packet.Write(_targetId);
        SendTCPData(_packet);
    }
}
```

Giving the packet its ID
Writing the data to the packet
Sending the packet via TCP

בדוגמה הזאת אנחנו משתמשים ב- using בעת יצירת הפקטה בשביל "לנקות" אותה אחרי השימוש. (המחלקה יורשת מ-IDisposable לכן דבר זה הכרחי) בנוסף הדרך שבה אנחנו נותנים לפקטה את ה-ID שלה היא בעזרת enum שמחזיק שמות של פקטות (ומתחיל מאחד) אנחנו עושים זאת בגלל שיותר נוח להבין מה כל פקטה עושה ככה.

אבל איפה הDataLength?!

לא שכחנו ממנו, למעשה בפעולה SendTCPData()
מוסיפה את האורך לתחילת הפקטה
על ידי שימוש בפעולה WriteLength()
של המחלקה Packet

```
/// <summary>Sends a packet to the server via TCP.</summary>
/// <param name="_packet">The packet to send to the sever.</param>
9 references
private static void SendTCPData(Packet _packet)
{
    _packet.WriteLength();
    Client.instance.tcp.SendData(_packet);
}
```

הפעולה WriteLength() של Packet

```
/// <summary>Inserts the length of the packet's content at the start of the buffer.</summary>
2 references
public void WriteLength()
{
    buffer.InsertRange(0, BitConverter.GetBytes(buffer.Count)); // Insert the byte length of the packet at the very beginning
}
```

אבל לכתוב את הפקטות כל אחד יכול עכשיו מגיע האתגר, קריאת הפקטות.

קריאה וטיפול בפקטה

*בגלל שהדגמנו על פקטה שנשלחת מהלקוח נדגים על טיפול בצד השרת

לפני שנתחיל לדבר על קריאה וטיפול בפקטה אני רוצה להסביר על delegate methods

Delegate method - נותנות לנו דרך להעביר פעולות כפרמטרים או לאחסן אותם (את הפעולות) במשתנים. דוגמה לDelegate:

```
public delegate void PacketHandler(int _fromClient, Packet _packet);
```

(הדוגמה נלקחה מהשרת למען ההסבר אך קיימת פעולה דומה גם בלקוח)

אז למה זה חשוב לנו?

גם בשרת וגם בלקוח קיים מילון בשם (packetHandlers) שלוקח מספרים שלמים (int) כמפתחות ו Delegets כערכים. הנה דוגמה למילון של השרת:

```
packetHandlers = new Dictionary<int, PacketHandler>()
{
    { (int)ClientPackets.welcomeReceived, ServerHandle.WelcomeReceived },
    { (int)ClientPackets.playerMovement, ServerHandle.PlayerMovement },
    { (int)ClientPackets.teleportToMap, ServerHandle.TeleportToMap },
    { (int)ClientPackets.getRoles, ServerHandle.GetRoles},
    { (int)ClientPackets.attemptKill,ServerHandle.AttemptKill},
    { (int)ClientPackets.updateTaskProgressServer, ServerHandle.UpdateTaskProgressServer},
    { (int)ClientPackets.callEmergencyMeeting, ServerHandle.CallEmergencyMeeting},
    { (int)ClientPackets.playerStartMadBayScan, ServerHandle.PlayerStartMadBayScan },
    { (int)ClientPackets.castVote, ServerHandle.CastVote},
    { (int)ClientPackets.gameStatus,ServerHandle.GameStatus}
};
```

בואו ניקח רגע כדוגמה את האיבר הראשן במילון. ניתן לראות שהמפח שלו הוא 1,
(בגלל שכמו שאמרנו קודם הEnum מתחיל מ1)
והערך שלו הוא קריאה לפעולה בשם ()WelcomeReceived של המחלקה ServerHandle.

איך אנו משתמשים בזה ואיך זה קשור לטיפול בפקטה?

בעצם המילון הזה משמש כמו "מרכזיה" לכל הפקטות. כאשר אנחנו מקבלים פקטה חדשה נוכל לקרוא ממנה את 4 הבייטים השניים ולקבל את הID שלה, ואם ניתן למילון הזה את הID של הפקטה הוא יוכל לקרוא לפעולה המתאימה לטיפול בפקטה (איזה יופי!)

אבל איפה כל זה קורה? איך יודעים אם מקבלים פקטה? ואיך מעבירים פרמטרים לפעולה?
בשרת כאשר אנחנו מקבלים פקטה אנחנו קוראים לפעולה HandleData של המחלקה TCP או UDP. (נסביר ונגדים על פקטת TCP aאם מבינים אותה קל להבין מה קורה לפקטת UDP)
בTCP הפעולה ארוכה אך אל דאגה נעבור עליה שלב שלב.

```
public class TCP
{
    public TcpClient socket;

    private readonly int id;
    private NetworkStream stream;
    private Packet receivedData;
    private byte[] receiveBuffer;
```

ראשית בוא נראה את תכונות המחלקה TCP
דבר ראשון יש לנו את הסוקט שלנו.
אחר כך את הID של הלקוח שחיבור הTCP שייך לו
ה stream שלנו
פקטה שמיצגת את המידע שקיבלנו
באפר שנשתמש בו כדי לאכסן את המידע בצורה זמנית.

עכשיו החלק הראשן של הפעולה(ראו קוד למטה). פה אנחנו מעדכנים את הבייטים של הפקטה receivedData
ובודקים האם קיבלנו פקטה חדשה אנו עושים זאת בעזרת הפעולה UnreadLength של Packet שנראת כך =>

```
public int UnreadLength()
{
    return Length() - readPos;
}
```

הפעולה Length מחזירה את אורך הבפר.

עכשיו אם גילינו שיש יותר מ4 בייטים שלא קראנו זה אומר שיש לנו פקטה שמחכה שנטפל בה (ה datalength הוא int ותופס 4 בייטים) עכשיו נקרא אותו בעזרת ()ReadInt של Packet
נבדוק שהפקטה לא ריקה ויש סיבה להמשיך לקרוא ומכאן נמשיך לחלק השני
החלק הראשון שעליו הרגע דיברנו נראה כך:

```
int _packetLength = 0;

receivedData.SetBytes(_data);
if (receivedData.UnreadLength() >= 4)
{
    // If clients received data contains a packet
    _packetLength = receivedData.ReadInt();
    if (_packetLength <= 0)
    {
        // If packet contains no data
        return true;
    }
}
```

updating packet's bits

Reading dataLength

<u>ועכשיו לחלק השני בקריאת הפקטה</u>

בחלק הזה אנחנו יוצרים פקטה חדשה וממלאים אותה במידע שאנחנו קוראים לפי אורך הפקטה אנחנו עושים זאת בעזרת הפעולה הבונה השלישית של Packet שנראת כך:

אנחנו עושים את זה בגלל שהDelegate שלנו צריך פקטה

```
/// <summary>Creates a packet from which data can be read. Used for receiving.</summary>
/// <param name="_data">The bytes to add to the packet.</param>
3 references
public Packet(byte[] _data)
{
    buffer = new List<byte>(); // Initialize buffer
    readPos = 0; // Set readPos to 0

    SetBytes(_data);
}
```

עכשיו שיש לנו פקטה אנחנו נקרא את 4 הביטים הבאים בעזרת ()ReadInt ארבעת ביטים האלו מסמנים את הPacketID ועכשיו הרגע שחיכינו לו. נקרא לפעולת הHandle המתאימה בעזרת המילון שלנו הדבר יראה ככה

Specifying which handle method to use — SenderID — The packet (without Data length & Packet ID)

```
Server.packetHandlers[_packetId](id, _packet); // Call appropriate method to handle the packet
```

זכרו שאנחנו קוראים למפעולות דרך ה Delegate שלנו וכמו שראינו בהתחלה ה Delagate מחכה ל2 פרמטרים והם:

_fromClient - שמסמל מי השחקן שממנו התקבלה הפקטה. במקרה הזה בפרמטר מועבר משנה בשם "id" שלא הזכרתי אותו עד עכשיו לכן ארחיב, המחלקה TCP והמלקה UDP נמצאות בתוך מחלקה אחרת שנקראת Client ואחת מתכונות הClient היא ה-id שאותו אנחנו מעבירים כאן.

_packet - פקטה... הפקטה שאנחנו מעבירים היא בשלב שבו כל המידע שיש בפקטה זה המידע שמענין אותנו (הורדנו ממנה את ה DataLenth ואת הPacketID).

את הפעולה שתיארנו עכשיו חשוב מאוד לבצע על ה MainThread של Unity בגלל שאם לא נעשה את זה עליו יכולות להיווצר בעיות סינכרון בין הלקוחות. איבוד נתונים ודברים רעים דומים. אנחנו מבצעים דברים על הMainThread בעזרת המחלקה ThreadManager (ראו קוד בסוף) אנחנו משתמשים ב"lambda expression" כדי להעביר פעולה לפעולה שתבצע את הפעולה על הMainThread. **החלק השני נראה כך:**

```
ThreadManager.ExecuteOnMainThread(() =>
{
    using (Packet _packet = new Packet(_packetBytes))
    {
        int _packetId = _packet.ReadInt();
        Server.packetHandlers[_packetId](id, _packet);
    }
});
```

לאחר מכן אנחנו בודקים אם יש לנו עוד פקטה ואם לא אז אנחנו מסיימים הדבר נראה כך:

```
_packetLength = 0; // Reset packet length
if (receivedData.UnreadLength() >= 4)
{
    // If client's received data contains another packet
    _packetLength = receivedData.ReadInt();
    if (_packetLength <= 0)
    {
        // If packet contains no data
        return true; // Reset receivedData instance to allow it to be reused
    }
}
```

<u>בסוף כל הפעולה נראת ככה:</u>

```
/// <summary>Prepares received data to be used by the appropriate packet handler methods.</summary>
/// <param name="_data">The recieved data.</param>
1 reference
private bool HandleData(byte[] _data)
{
    int _packetLength = 0;

    receivedData.SetBytes(_data);
    if (receivedData.UnreadLength() >= 4)
    {
        // If clients received data contains a packet
        _packetLength = receivedData.ReadInt();
        Debug.Log("PACKETLENGTH = " + _packetLength);
        if (_packetLength <= 0)
        {
            // If packet contains no data
            return true;
        }
    }

    while (_packetLength > 0 && _packetLength <= receivedData.UnreadLength())
    {
        // While packet contains data AND packet data length doesn't exceed the length of the packet we are reading
        byte[] _packetBytes = receivedData.ReadBytes(_packetLength);
        ThreadManager.ExecuteOnMainThread(() =>
        {
            using (Packet _packet = new Packet(_packetBytes))
            {
                int _packetId = _packet.ReadInt();
                Server.packetHandlers[_packetId](id, _packet); // Call appropriate method to handle the packet
            }
        });

        _packetLength = 0; // Reset packet length
        if (receivedData.UnreadLength() >= 4)
        {
            // If client's received data contains another packet
            _packetLength = receivedData.ReadInt();
            if (_packetLength <= 0)
            {
                // If packet contains no data
                return true; // Reset receivedData instance to allow it to be reused
            }
        }
    }

    if (_packetLength <= 1)
    {
        return true; // Reset receivedData instance to allow it to be reused
    }

    return false;
}
```

אתם בטח שואלים למה אנחנו מחזירים true בחלקים מהפעולה. אנחנו עושים זאת כי קריאת הפעולה מבוצעת כפרמטר לפעולה Reset שלך המחלקה Packet
הקריאה נראת כך:

```
receivedData.Reset(HandleData(_data)); // Reset receivedData if all data was handled
```

והפעולה Reset נראת כך

```
/// <summary>Resets the packet to allow it to be reused.
1 reference
public void Reset(bool _shouldReset = true)
{
    if (_shouldReset)
    {
        buffer.Clear(); // Clear buffer
        readableBuffer = null;
        readPos = 0; // Reset readPos
    }
    else
    {
        readPos -= 4; // "Unread" the last read int
    }
}
```

<u>אז הבנו איך מבינים לאן הפקטה הולכת עכשיו לחלק האחרון: טיפול במידע שקיבלנו:</u>
נגדים על אותה בקטה שבנינו קודם. (תזכורת: הפקטה הכילה מספר שמייצג את השחקן שאנחנו רוצים להרוג) אז בצד של השרת נקראה הפעולה AttemptKill של ServerHandle ראשית נקרא מהפקטה שקיבלנו את המידע המצופה במקרה שלנו int אז נשתמש בפעולה ReadInt של Packet ועכשיו שיש לנו אותו אנחנו יכולים פשוט להתשמש במידע. הפעולה נראת כך:

```
public static void AttemptKill(int _fromClient, Packet _packet)
{
    int _targetedId = _packet.ReadInt();
    Debug.Log("attempted to kill player: "+_targetedId);
    if (!Server.clients[_targetedId].player.isImpostor)
    {
        Server.clients[_targetedId].player.isAlive = false;
        Debug.Log("player: " + _targetedId + " has been killed");

        ServerSend.EliminatePlayer(_targetedId);

    }
}
```

Reading an int from the packet

Letting the players know of the changes
(sending a packet im almost the same way as before)

Using the data

בסוף הפעולה אנחנו שולחים פקטה חזרה ללקוחות (לכולם) הפקטה נבנת כמו בלקוח.

Thread manager

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

Unity Script (1 asset reference) | 3 references
public class ThreadManager : MonoBehaviour
{
    private static readonly List<Action> executeOnMainThread = new List<Action>();
    private static readonly List<Action> executeCopiedOnMainThread = new List<Action>();
    private static bool actionToExecuteOnMainThread = false;

    Unity Message | 0 references
    private void FixedUpdate()
    {
        UpdateMain();
    }

    /// <summary>Sets an action to be executed on the main thread.</summary>
    /// <param name="action">The action to be executed on the main thread.</param>
    3 references
    public static void ExecuteOnMainThread(Action action)

    {
        if (action == null)
        {
            Console.WriteLine("No action to execute on main thread!");
            return;
        }

        lock (executeOnMainThread)
        {
            executeOnMainThread.Add(action);
            actionToExecuteOnMainThread = true;
        }
    }

    /// <summary>Executes all code meant to run on the main thread. Called ONLY from the main thread.</summary>
    1 reference
    public static void UpdateMain()
    {
        if (actionToExecuteOnMainThread)
        {
            executeCopiedOnMainThread.Clear();
            lock (executeOnMainThread)
            {
                executeCopiedOnMainThread.AddRange(executeOnMainThread);
                executeOnMainThread.Clear();
                actionToExecuteOnMainThread = false;
            }

            for (int i = 0; i < executeCopiedOnMainThread.Count; i++)
            {
                executeCopiedOnMainThread[i]();
            }
        }
    }
}
```

במחלקה הזאת נעזרתי במישהו רנדומלי מפלטפורמה בשם Discord
הסברתי לו את הבעיות שיש לי בקוד (מה שדיברתי עליו קודם "בעיות סינכרון בין הלקוחות. איבוד נתונים ועוד") והוא הביא לי את הקוד הזה.

בנוסף במהלך הפרויקט נעזרתי ב:StackOverflow, Unity Forums, Discord,ChatGPT