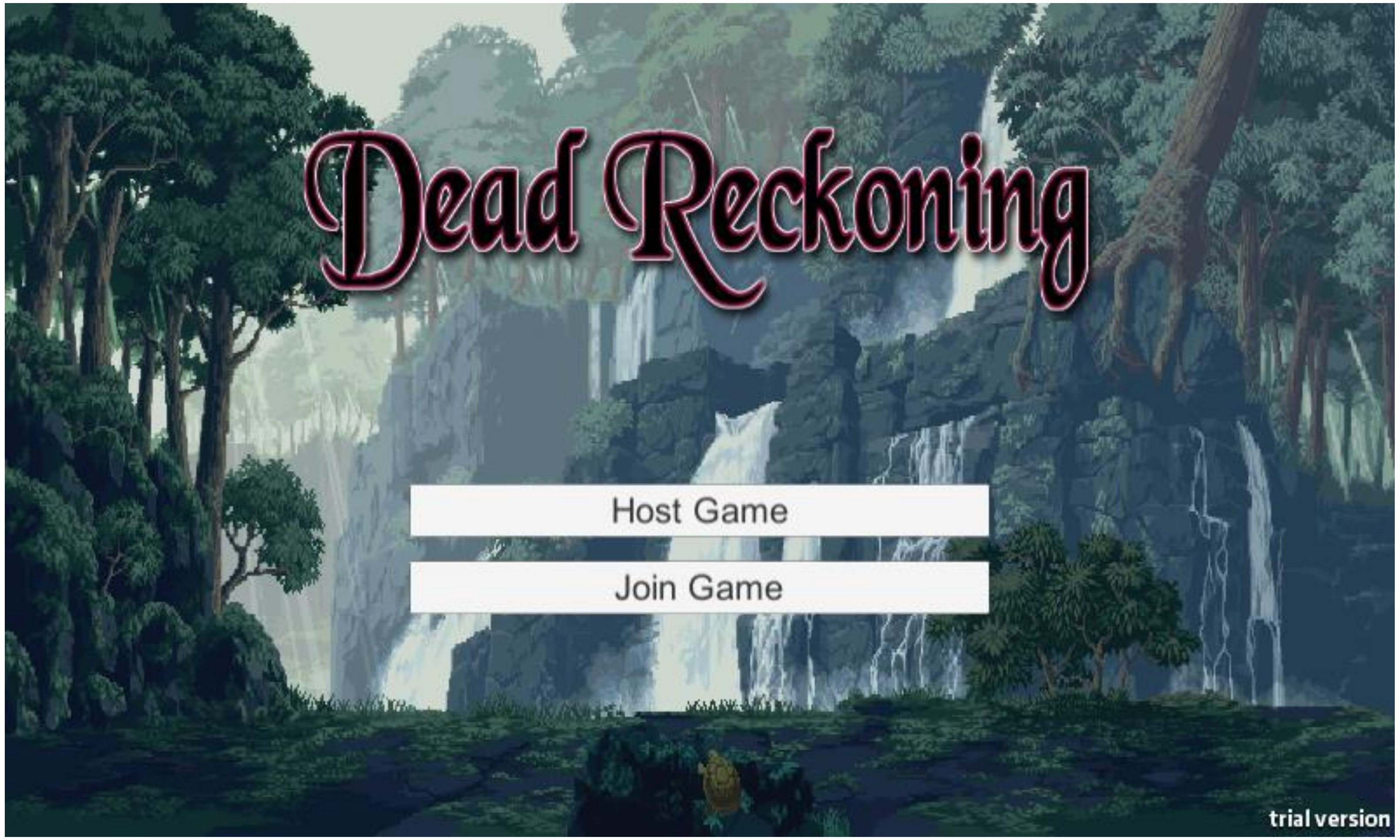
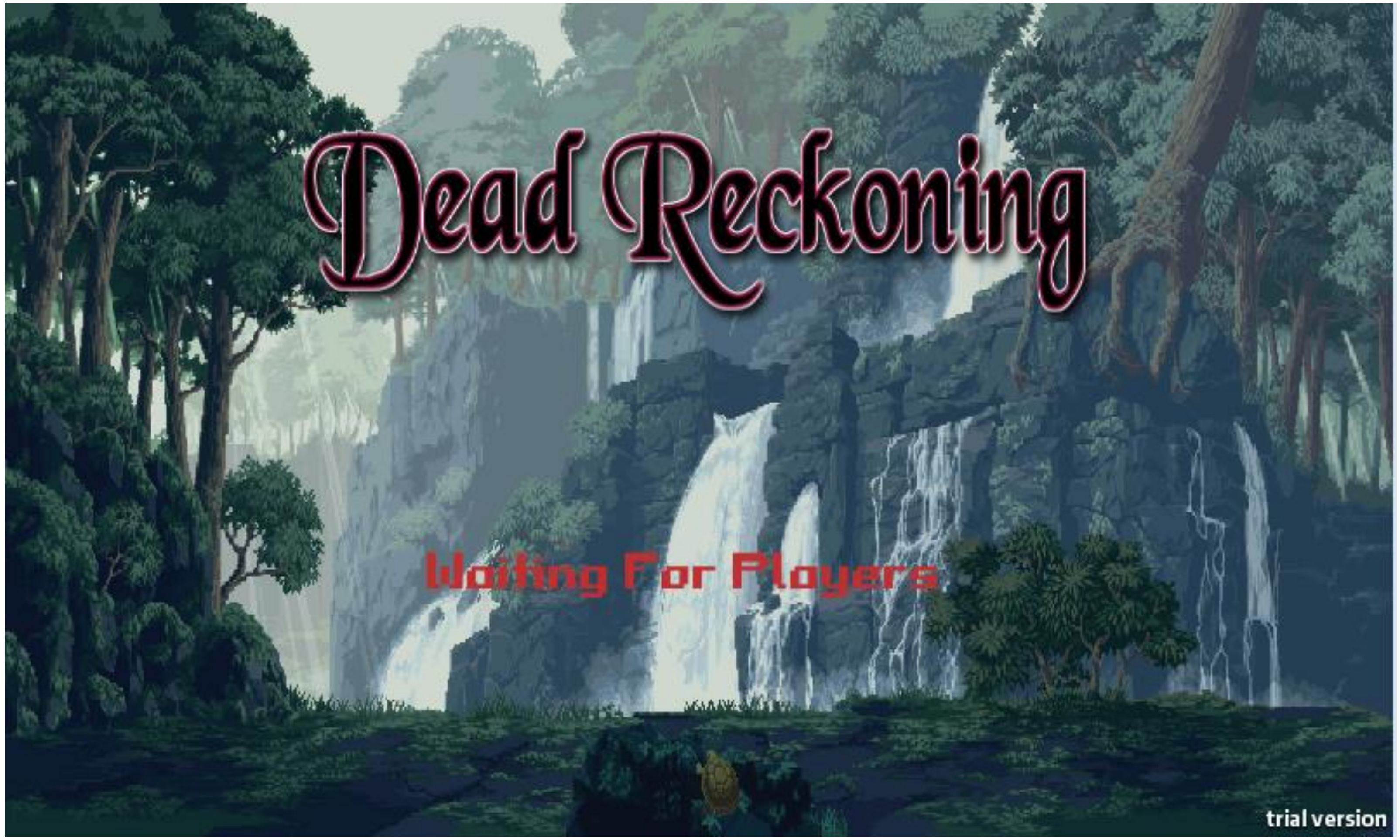
Dead Reckoning

By Janek Uchman

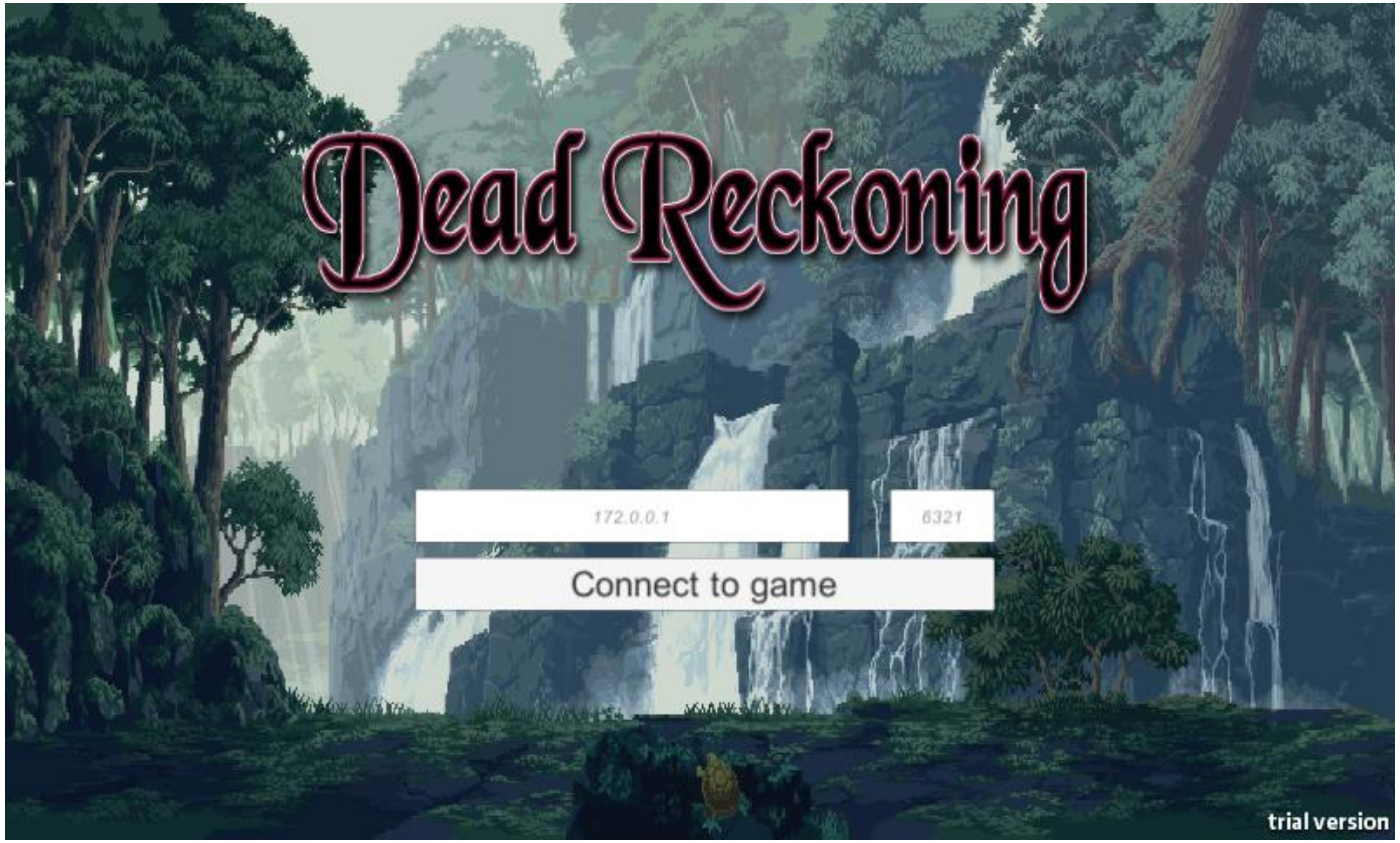
A brief run through of the game



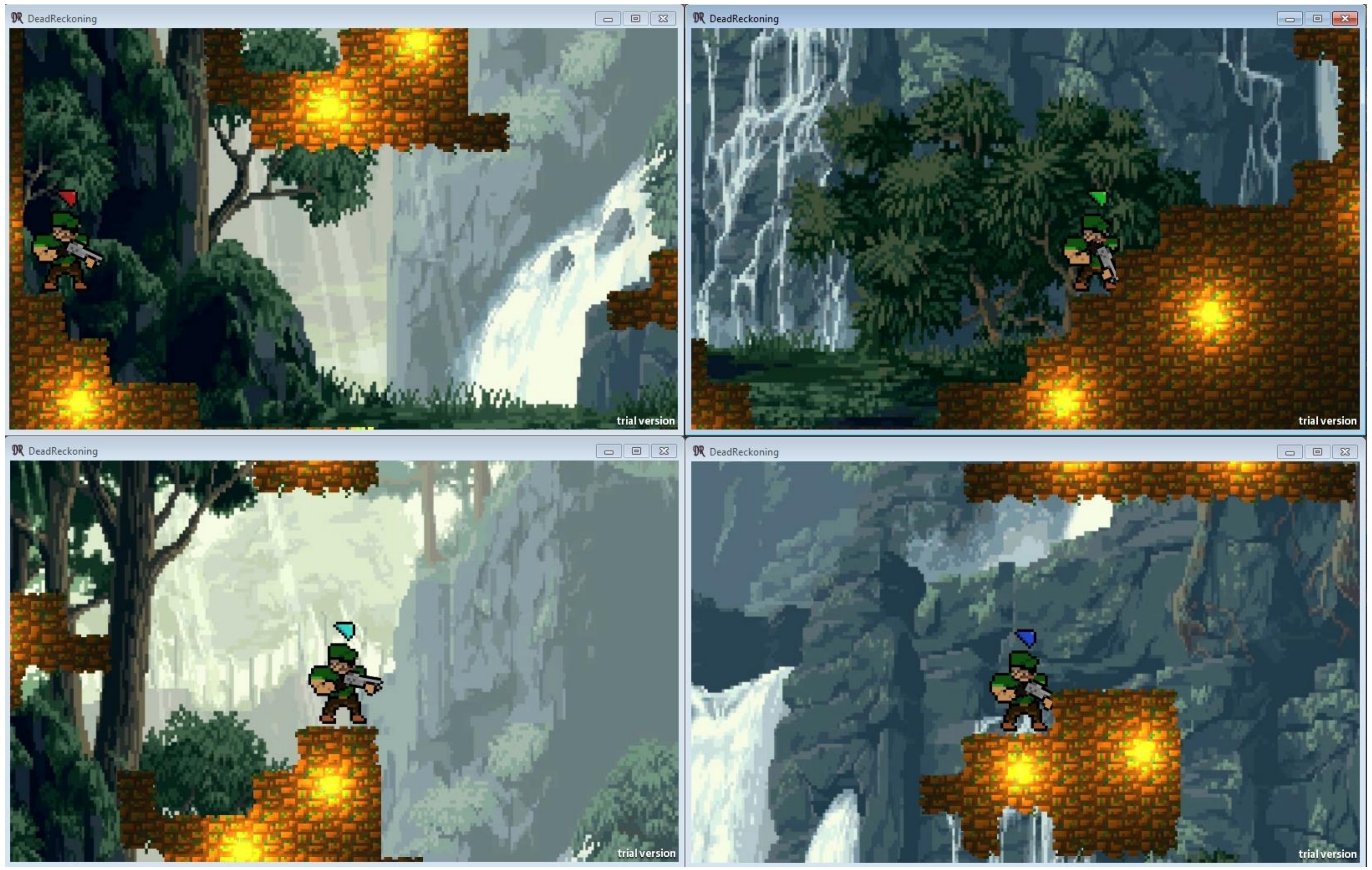
A brief run through of the game



A brief run through of the game



A brief run through of the game



A brief run through of the game

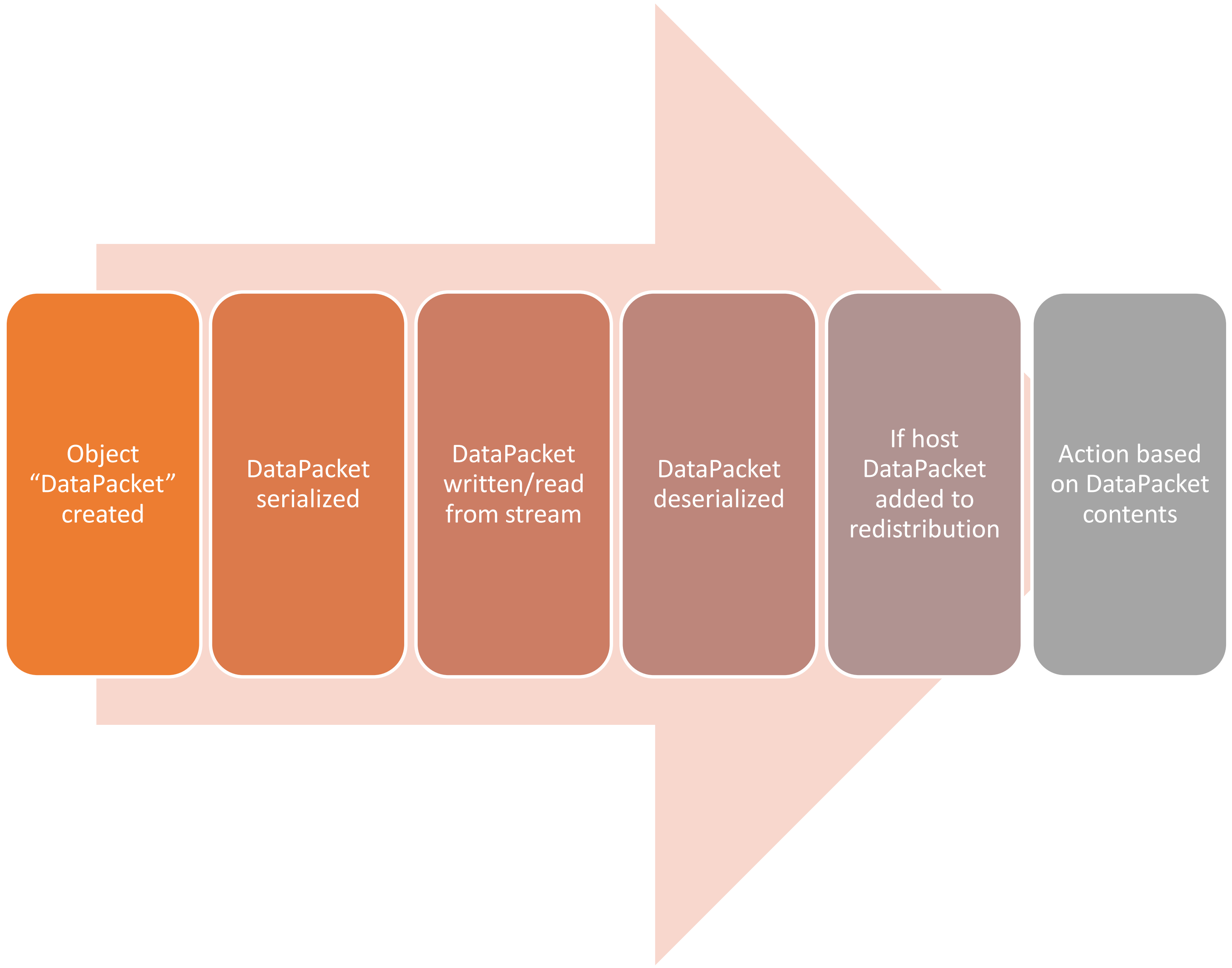
A brief run through of the game

- Enemy players controlled through component
- Player has separate components
- Additional players handled client side

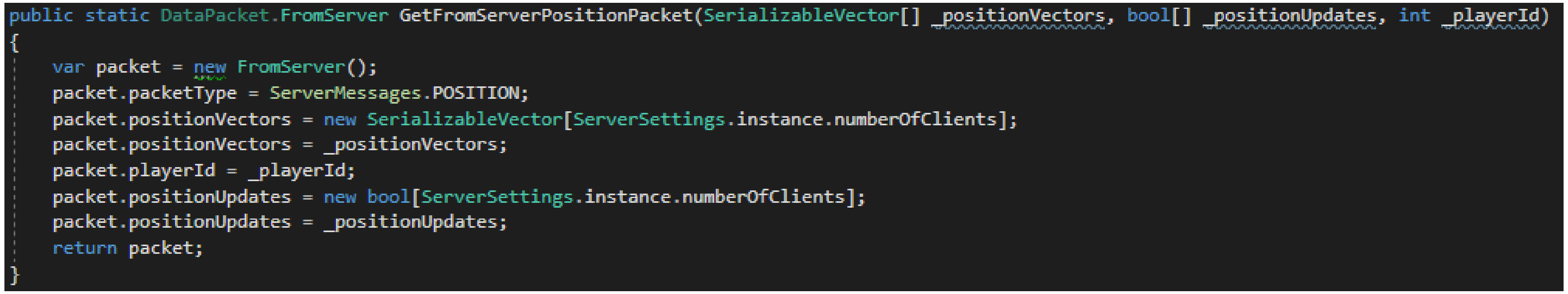
Networking layout

- .NET Sockets/TCP
- Server-Client
- Async using select and Unity coroutines

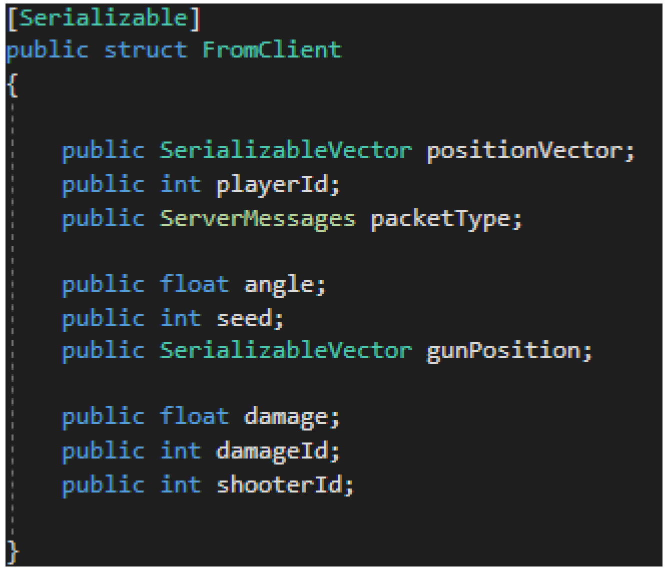
Packet Pipeline


Object "DataPacket" created
DataPacket serialized
DataPacket written/read from stream
DataPacket deserialized
If host DataPacket added to redistribution
Action based on DataPacket contents

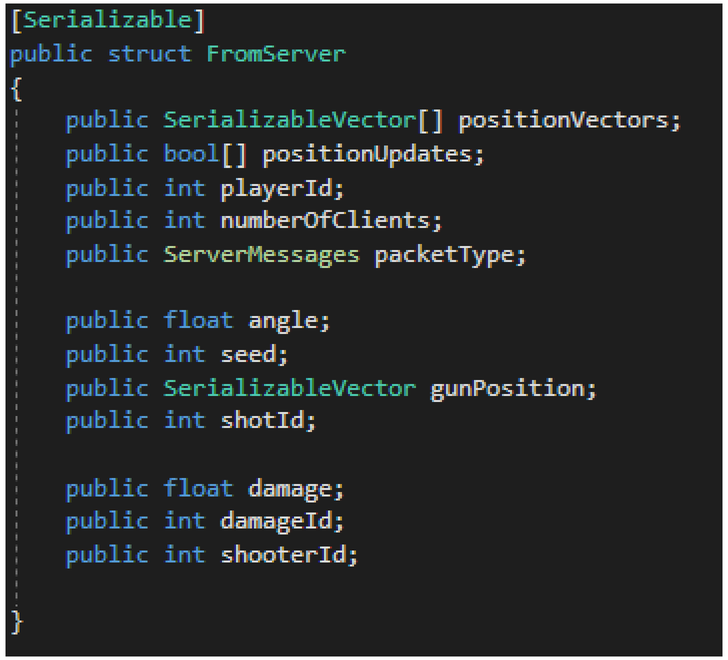
DataPacket



```
public static DataPacket.FromServer GetFromServerPositionPacket(SerializableVector[] _positionVectors, bool[] _positionUpdates, int _playerId)
{
    var packet = new FromServer();
    packet.packetType = ServerMessages.POSITION;
    packet.positionVectors = new SerializableVector[ServerSettings.instance.numberOfClients];
    packet.positionVectors = _positionVectors;
    packet.playerId = _playerId;
    packet.positionUpdates = new bool[ServerSettings.instance.numberOfClients];
    packet.positionUpdates = _positionUpdates;
    return packet;
}
```

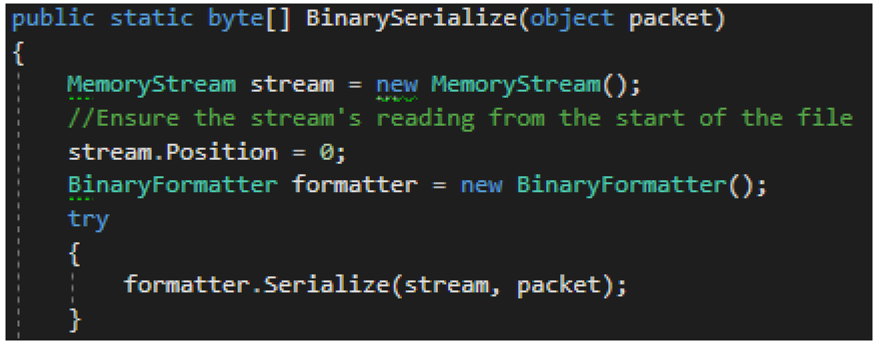


```
[Serializable]
public struct FromClient
{

    public SerializableVector positionVector;
    public int playerId;
    public ServerMessages packetType;

    public float angle;
    public int seed;
    public SerializableVector gunPosition;

    public float damage;
    public int damageId;
    public int shooterId;

}
```

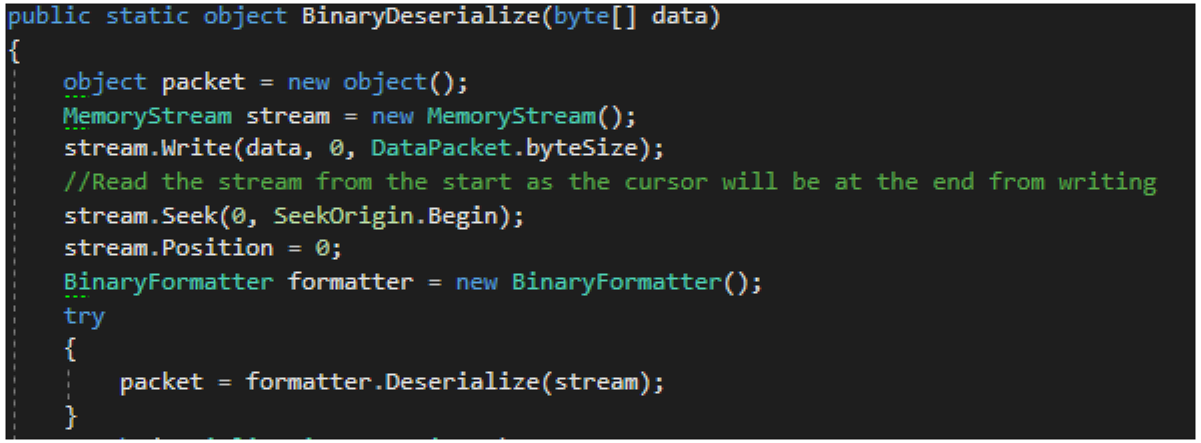


```
[Serializable]
public struct FromServer
{
    public SerializableVector[] positionVectors;
    public bool[] positionUpdates;
    public int playerId;
    public int numberOfClients;
    public ServerMessages packetType;

    public float angle;
    public int seed;
    public SerializableVector gunPosition;
    public int shotId;

    public float damage;
    public int damageId;
    public int shooterId;

}
```

Serialization

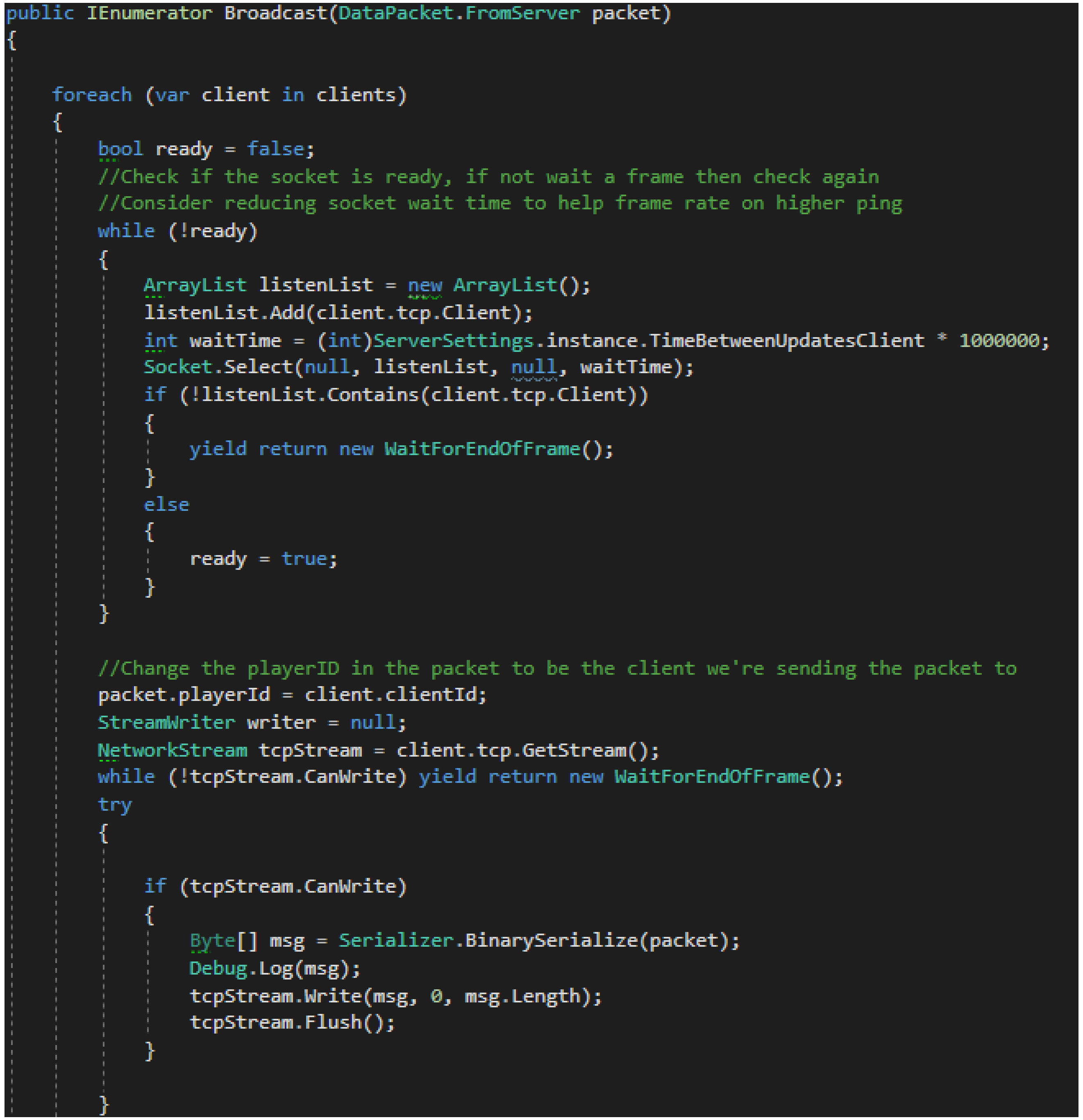


```
public static byte[] BinarySerialize(object packet)
{
    MemoryStream stream = new MemoryStream();
    //Ensure the stream's reading from the start of the file
    stream.Position = 0;
    BinaryFormatter formatter = new BinaryFormatter();
    try
    {
        formatter.Serialize(stream, packet);
    }
```



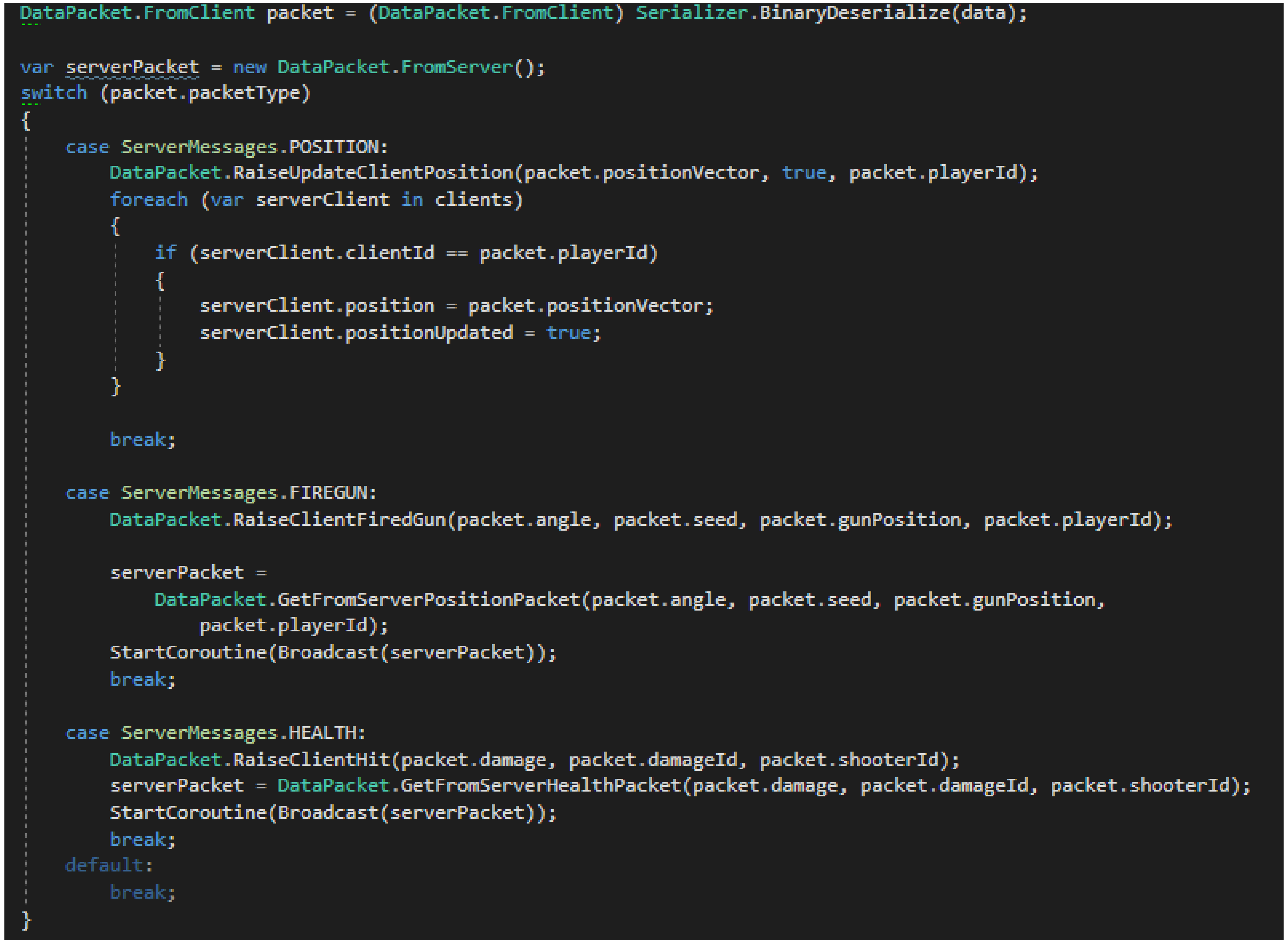
```
public static object BinaryDeserialize(byte[] data)
{
    object packet = new object();
    MemoryStream stream = new MemoryStream();
    stream.Write(data, 0, DataPacket.byteSize);
    //Read the stream from the start as the cursor will be at the end from writing
    stream.Seek(0, SeekOrigin.Begin);
    stream.Position = 0;
    BinaryFormatter formatter = new BinaryFormatter();
    try
    {
        packet = formatter.Deserialize(stream);
    }
```

Distribution



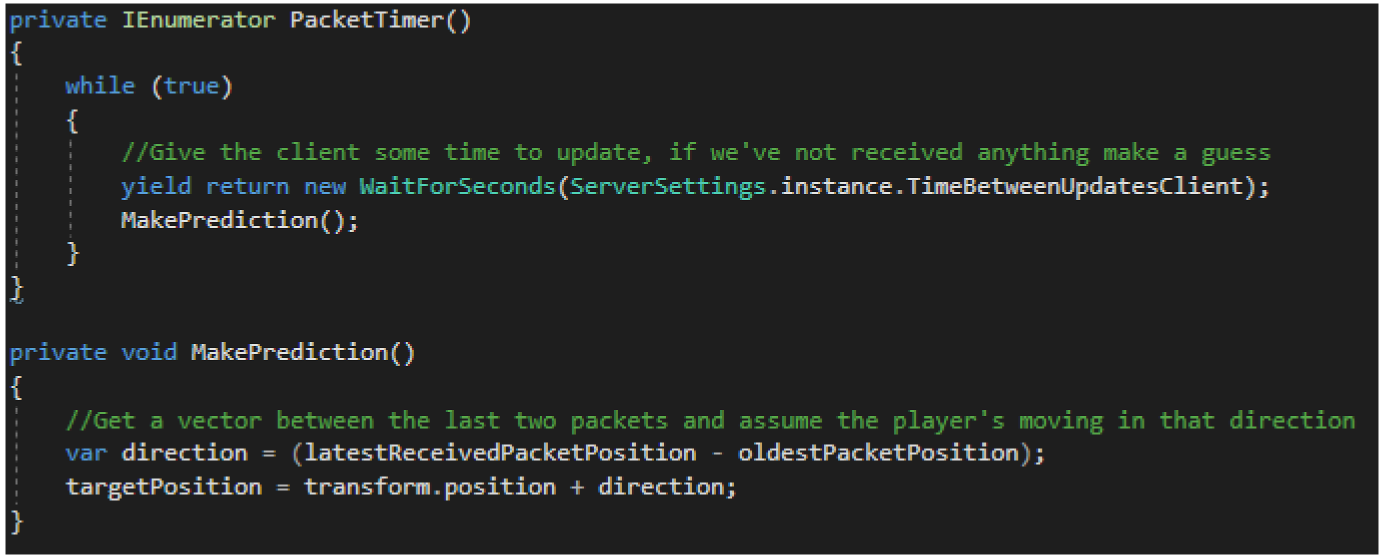
```
public IEnumerator Broadcast(DataPacket.FromServer packet)
{

    foreach (var client in clients)
    {
        bool ready = false;
        //Check if the socket is ready, if not wait a frame then check again
        //Consider reducing socket wait time to help frame rate on higher ping
        while (!ready)
        {
            ArrayList listenList = new ArrayList();
            listenList.Add(client.tcp.Client);
            int waitTime = (int)ServerSettings.instance.TimeBetweenUpdatesClient * 1000000;
            Socket.Select(null, listenList, null, waitTime);
            if (!listenList.Contains(client.tcp.Client))
            {
                yield return new WaitForEndOfFrame();
            }
            else
            {
                ready = true;
            }
        }

        //Change the playerID in the packet to be the client we're sending the packet to
        packet.playerId = client.clientId;
        StreamWriter writer = null;
        NetworkStream tcpStream = client.tcp.GetStream();
        while (!tcpStream.CanWrite) yield return new WaitForEndOfFrame();
        try
        {

            if (tcpStream.CanWrite)
            {
                Byte[] msg = Serializer.BinarySerialize(packet);
                Debug.Log(msg);
                tcpStream.Write(msg, 0, msg.Length);
                tcpStream.Flush();
            }

        }
```

Handling the packet



```
DataPacket.FromClient packet = (DataPacket.FromClient) Serializer.BinaryDeserialize(data);

var serverPacket = new DataPacket.FromServer();
switch (packet.packetType)
{
    case ServerMessages.POSITION:
        DataPacket.RaiseUpdateClientPosition(packet.positionVector, true, packet.playerId);
        foreach (var serverClient in clients)
        {
            if (serverClient.clientId == packet.playerId)
            {
                serverClient.position = packet.positionVector;
                serverClient.positionUpdated = true;
            }
        }

        break;

    case ServerMessages.FIREGUN:
        DataPacket.RaiseClientFiredGun(packet.angle, packet.seed, packet.gunPosition, packet.playerId);

        serverPacket =
            DataPacket.GetFromServerPositionPacket(packet.angle, packet.seed, packet.gunPosition,
                packet.playerId);
        StartCoroutine(Broadcast(serverPacket));
        break;

    case ServerMessages.HEALTH:
        DataPacket.RaiseClientHit(packet.damage, packet.damageId, packet.shooterId);
        serverPacket = DataPacket.GetFromServerHealthPacket(packet.damage, packet.damageId, packet.shooterId);
        StartCoroutine(Broadcast(serverPacket));
        break;
    default:
        break;
}
```

Prediction



```
private IEnumerator PacketTimer()
{
    while (true)
    {
        //Give the client some time to update, if we've not received anything make a guess
        yield return new WaitForSeconds(ServerSettings.instance.TimeBetweenUpdatesClient);
        MakePrediction();
    }
}

private void MakePrediction()
{
    //Get a vector between the last two packets and assume the player's moving in that direction
    var direction = (latestReceivedPacketPosition - oldestPacketPosition);
    targetPosition = transform.position + direction;
}
```

Downfalls

- Skating
- Client side hit detection
- UDP for positional updates

Occasional serialization packet loss solutions

Asynchronous Send/Receive

Send everything as one packet

Different serialization methods

Dead Reckoning