

Multiplayer Unity Game Engine.

(Using Dotnetty network framework)

Table of Contents

Table of Contents	2
Introduction	3
Project Installation	7
Tutorial	9
Client Input Controls	13
Data Structures & Algorithms	14
Server Framework Design	18
Server Systems And Gameplay	23
Server Classes	26
Client Framework Design	35
Client Classes	38
Project Specifications	46
Conclusion	48
Libraries and References Used	50

Introduction

Games have always been a long passion of mine. Not only was my early childhood spent around finding ways to get games for free through ROMs. Which peaked my interest, on how that was possible. This interest ended up taking me to practicing game development on a public scale. Where i deployed live applications generating revenue to pay team members appropriately. It was much smaller of a scale in terms of audience but still live deployable nonetheless with thousands of plays. For this senior project I wanted to make a small game with little theme to it because assets are hard to come by, and without proper time for appropriate level designs a multiplayer game is basically pointless.

This small game I built is done on the Unity Game engine and comes with a full network framework to support up to 3 tested players on a server hosted on my own network public to other networks. I've included a player with a proper character controller that replicates to every other client. An enemy which also uses the character control methods to move a monster. I have a small combat system for a player vs npc. There is a dialogue system that is also part of the quest system I have in place. You can find weapons scattered on the map and can equip them as well, all doing different damage, You get a small 9 slot inventory space for active hotkeys which is also used as your saved inventory data. Which leads into the player account saving which tracks inventory items,

equipped items and other player stats like health and level. Knowing full well this is still incomplete of a game, the level of difficulty is mid tier. The added performance gains on the entire framework has added more value to the project in terms of knowledge gained. This network framework was designed with all the skills I have learned over the years working with many game engines, and a background is Computer Science.

The networking portion of the game gave me a lot of early complications. The initial setup was pretty easy to get going. I used DotNetty Library which is a java port for c# basically. This library lets me easily set up an asynchronous network between client and server. This allowed for me to have no yielding functions within the game thread for any given player at any given time. What gave me the most trouble was mainly sending a full packet to every client. When I first started the project I was assuming it was going smooth till I tested with 2 clients and saw that I lost important packet data. There was error being so i knew that there some stack tracing to be done. I spent over 3 weeks stuck on this trying to figure out why, and soon realized that I had to simply create a copy of the buffer for each client. I ended up doing 4 different network designs before finally reaching the one I wanted to submit with.

The whole character code design uses polymorphism design. We have characters define similar base values the both inherited classes will also use,

From there we derive a npc and player from a character. With a character I have assigned combat controllers and movement controllers. Both a player and an npc used these classes to drive based on unique input or forced input for npcs. I also use a npc to make a dialogue quest npc, which is treated as a non attackable Npc.

Testing phases for this project were done on my actual Ip and opened up to some friends. Only 2 or 3. I also tested on a local network, and as you would expect it was much smoother but i was receiving some lag for my movement. Personally i think it's because i didn't include a local player movement controller for the client to move first locally before getting the movement data from the server. I don't think it was a latency issue but it was still really good for the state I had it.

In conclusion, I'm well aware this is far from a complete game, but I know full well that a complete game cannot be done within 5 months and 1 man. Maybe if i stole it from somewhere and changed it up but that's way more demotivating and probably not worth the trouble of getting caught. I came in knowing I wanted to make a network based project and it had to be game development related because that's where i want to see my career going. I believe I achieved everything I set out to when trying to make this game. I have a better understanding of the unity engine. Along with a better understanding of the networking behind all games.

Multiplayer games on the unity engine normally run on the unity servers which means you'd have to pay a couple thousand dollars a year for an x amount of playerbase. With my project it would be free of cost as far as hosting goes. There's obviously unity Profit fees if marketed. Not saying mine is better than unitys cause it's not, but it's definitely a good resource to have under a resume.

Project Installation

Both client and server run on Version 2019.4.8f of the Unity Engine. Once you've downloaded the engine: <https://unity3d.com/get-unity/download>

Download the version package:

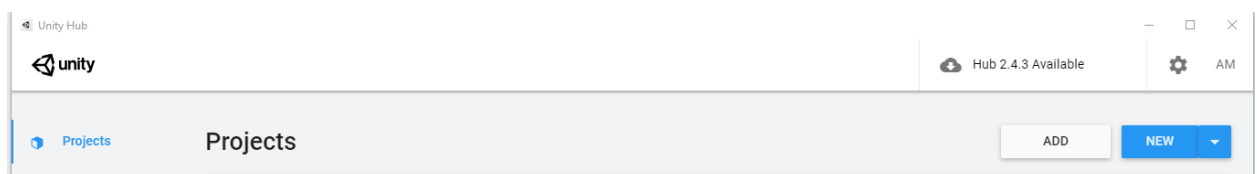
<https://unity3d.com/unity/whats-new/2019.4.8>

Clone the repository or extra the zip file onto your desktop.

<https://github.com/amandania/Senior-Project>

You will notice there is a client folder and a server folder. Both folders are independent project folders for unity to load. Make sure you have a unity license which is 100% free.

You must open the Unity dashbash and add the project directory into the dashboard. You will need to set the unity version here as well. Make sure you have that downloaded before you add the project to the dashboard.



Click Add and load the server and client individually.

Project Name	Unity Version	Target Platform	Last Modified	↑	🔍
Client C:\Users\Admin\Desktop\Senior-Project\Client Unity Version: 2019.4.8f1	2019.4.8f1 ▼	Current platform ▼	15 hours ago		⋮
Server C:\Users\Admin\Desktop\Senior-Project\Server Unity Version: 2019.4.8f1	2019.4.8f1 ▼	Current platform ▼	15 hours ago		⋮

Once you load them make sure you have unity version set to 2018.4.8f1

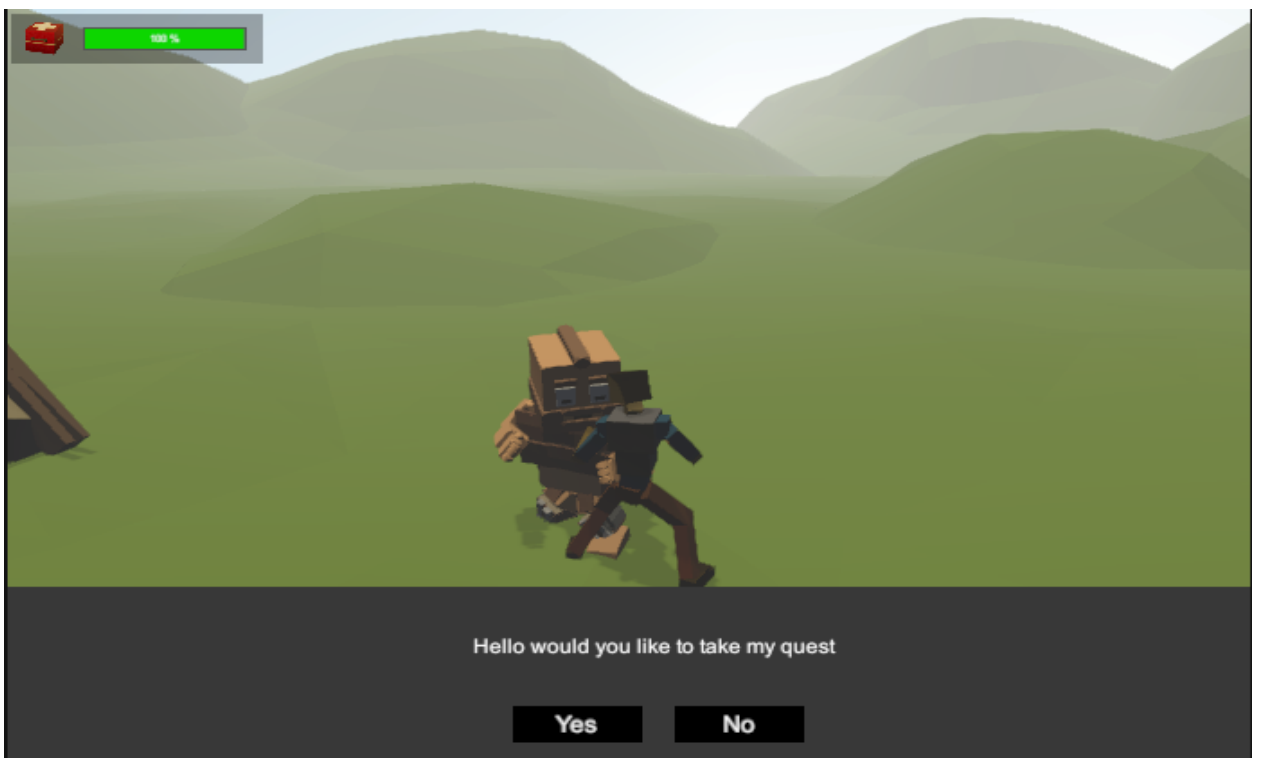
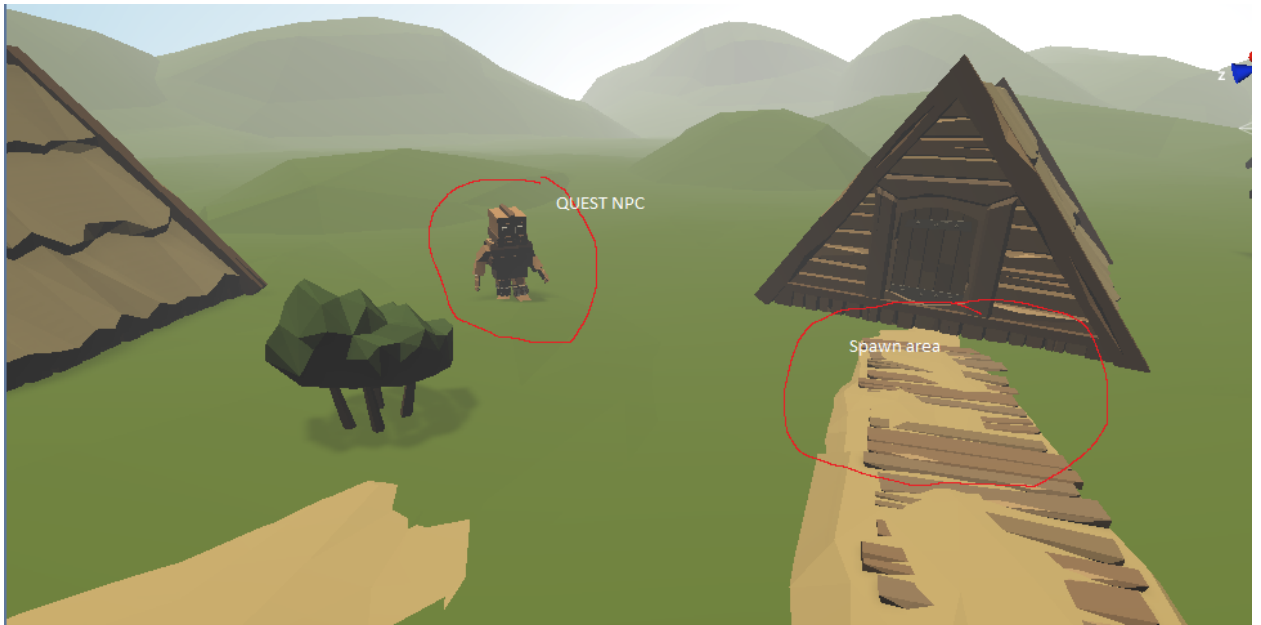
Once this is done you can either run the client from the engine or build an output version. Which is a launchable version of the client.

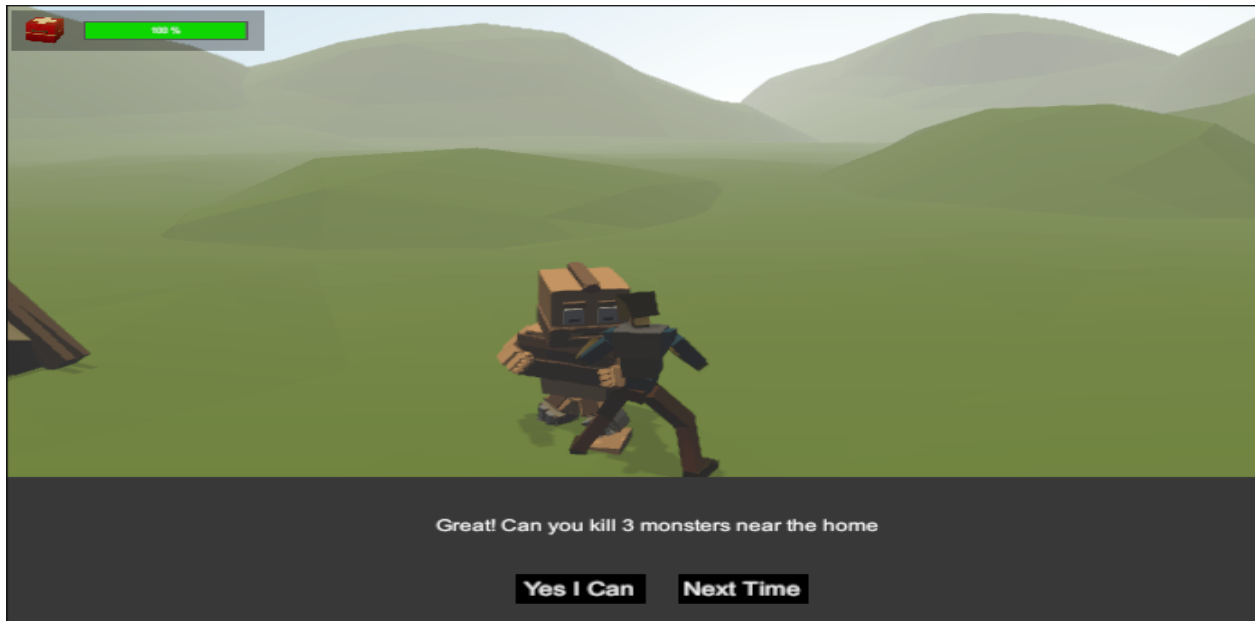
I've included a deployable version in the download

Tutorial

There is not too much to this game to learn, aside from actual gameplay content

You can find your first quest near the login spot. Walk up to it and press F.

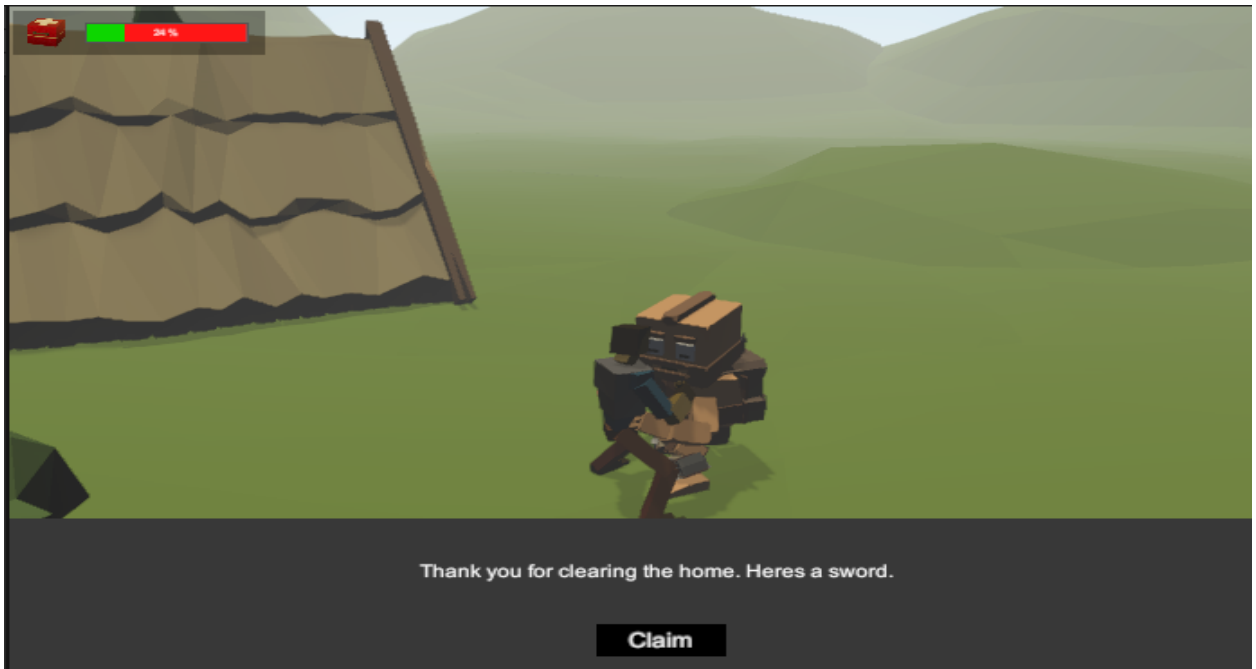




You will be tasked to kill 3 Bandits near the home.



After completing the quest you can then head back to the login spot and hand in your quest for a sword.



The sword gives you new attack speeds and attack animations at the cost of a smaller hitbox

You can also search for weapons throughout the map. Pick them up using the F key.



When you pick up weapons they will be saved and to use them, you press 1-9 from left to right.



Client Input Controls

Gameplay controls consist of Keyboard input only

ESCAPE Key - Logout screen toggled

Mouse Button 1 (left click) - is combat swing

WASD Keys - Are used for movement

Top number keys 1-9 are used for activate hotkey slots

Hold Mouse Button 2 (right click) - Will active strafing mode

Data Structures & Algorithms

HashMaps (*Dictionaries*)

We use this data in a lot of areas throughout the project. Anywhere we need key value mapping. For instance our equipment system has a key value pair using string for key and game object as value

```
//Map of transform name and current equipmodel
public Dictionary<Transform, GameObject> ItemToTransformMap = new Dictionary<Transform, GameObject>();

//Map of loadedTransform names;
public Dictionary<string, Transform> TransformMap = new Dictionary<string, Transform>();
```

They are also used store a list our possible quests

```
public Dictionary<string, Quest> PossibleQuests = new Dictionary<string, Quest>();
```

Also for some player quest related data.

```
public Dictionary<string, Quest> PlayerQuests { get; set; } = new Dictionary<string, Quest>();
```

Lists And Arrays

Store Hotkey items

Store List of started quests and their status

Various other areas of the project as well because Lists are pretty important for a lot of reasons.

```
public class Container
{
    public List<SlotItem> ContainerItems { get; set; }
```

Structures

Slot Item

PlayerSave

- Json Serialize and deserialize class
- Used for player saving and player loading

```
public struct PlayerSave {  
    public string Username { get; set; }  
    public string Password { get; set; }  
    public int PlayerLevel { get; set; }  
    public int CurrentHealth { get; set; }  
    public int MaxHealth { get; set; }  
    public SlotItem[] HotkeyItems { get; set; }  
    public Quest[] PlayerQuests { get; set; }  
  
    public int CurrentSlotEquipped { get; set; }  
}
```

Objects

Abstract object oriented programming behind some of our system design

Character Class

Npc : Character

Player : Character

Container

Hotkey : Container

BreathFirstSearch

- Used to find the descendant child by name

```

// Returns Descendant gameobject, returns
public Transform FindDeepChild(string a_childName)
{
    Queue<Transform> queue = new Queue<Transform>();
    queue.Enqueue(transform);
    while (queue.Count > 0)
    {
        var c = queue.Dequeue();
        if (c.name == a_childName)
            return c;
        foreach (Transform t in c)
            queue.Enqueue(t);
    }
    return null;
}

```

A* Pathfinding (deprecated Code) Uses Heap

- During pathfinding we create heaps of nodes to go through. We sort them by distance as well. To attempt get the shortest path

```

public PathResult FindPath(Vector3 startPos, Vector3 targetPos)
{
    Vector3[] waypoints = new Vector3[0];
    bool pathSuccess = false;

    Node startNode = Grid.NodeFromWorldPoint(startPos);
    Node targetNode = Grid.NodeFromWorldPoint(targetPos);

    startNode.m_parent = startNode;
}

```



```

if (startNode.m_walkable && targetNode.m_walkable)
{
    CustomHeap<Node> openSet = new CustomHeap<Node>(Grid.MaxSize);
    HashSet<Node> closedSet = new HashSet<Node>();

    openSet.Add(startNode);

    while (openSet.Count > 0)
    {
        Node currentNode = openSet.RemoveFirst();
        closedSet.Add(currentNode);

        if (currentNode == targetNode)
        {
            //print ("Path found: " + sw.ElapsedMilliseconds + " ms");
            pathSuccess = true;
            break;
        }

        foreach (Node neighbour in Grid.GetNeighbours(currentNode))
        {
            if (!neighbour.m_walkable || closedSet.Contains(neighbour))
            {
                continue;
            }

            int newMovementCostToNeighbour = currentNode.m_gCost + GetDistance(currentNode, neighbour) + neighbour.m_movementPenalty;
            if (newMovementCostToNeighbour < neighbour.m_gCost || !openSet.Contains(neighbour))
            {
                neighbour.m_gCost = newMovementCostToNeighbour;
                neighbour.m_hCost = GetDistance(neighbour, targetNode);
                neighbour.m_parent = currentNode;

                if (!openSet.Contains(neighbour))
                    openSet.Add(neighbour);
                else
                    openSet.UpdateItem(neighbour);
            }
        }
    }
}

```

Server Framework Design

The server uses the DotNetty Library to implement our network channels and session handling for data sending.

<https://github.com/Azure/DotNetty>

We also use Auto fac library to build all the dependencies the game world is dependent on. We treat these dependencies as pointer instances to either be used as a signal instance or for each construction of a class that is using the container. It was an attempt to implement an IOC framework. I later found out this was a bad idea as it caused thread issues which were solved by using UnityMainThread dispatcher. It's part of a Unity Library microsoft published as open source code.

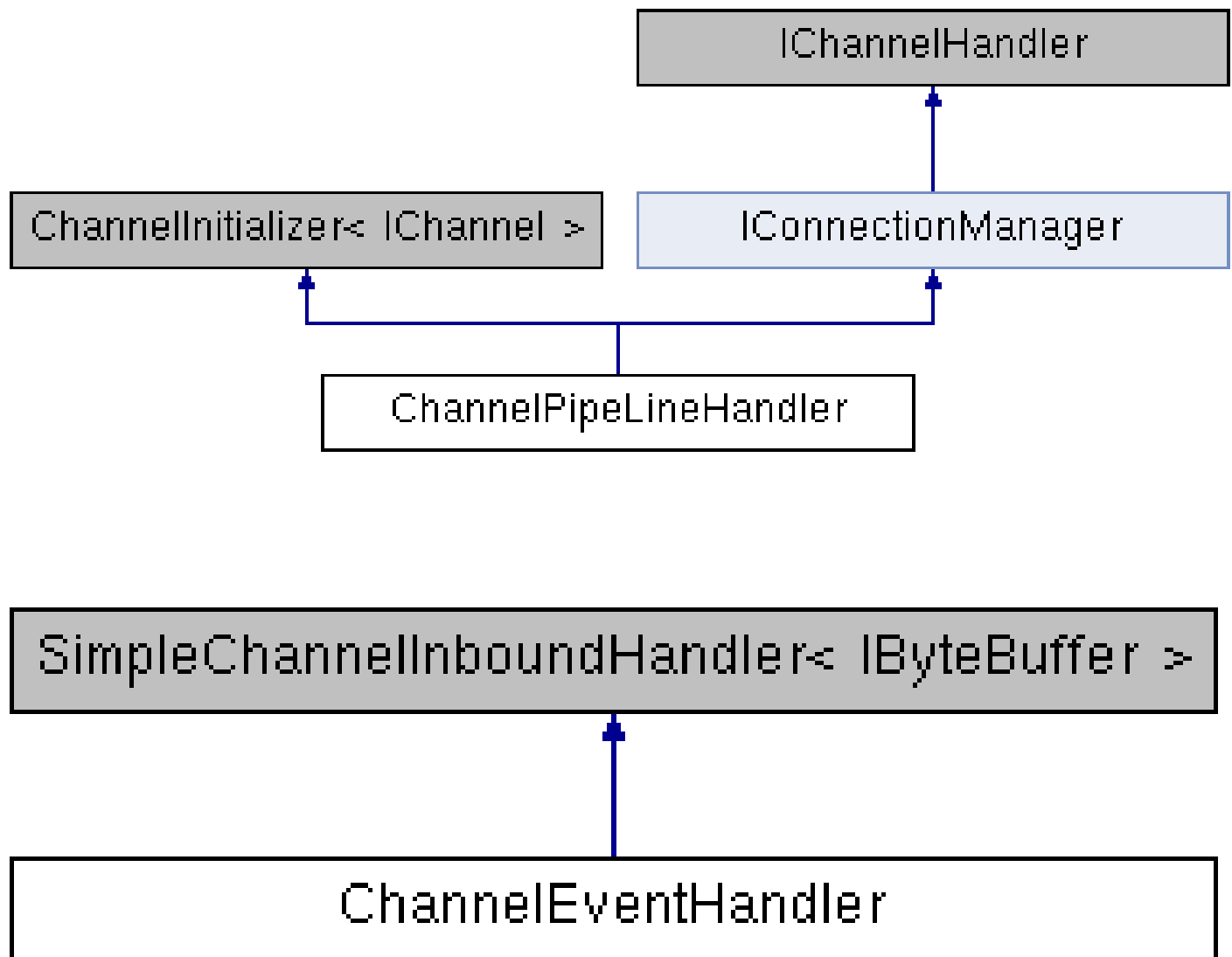
Full documentation on auto fac library can be found @

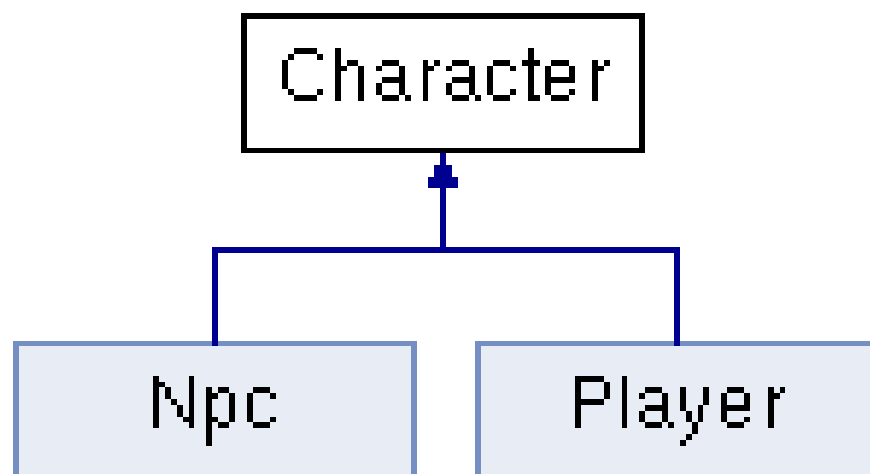
<https://autofaccn.readthedocs.io/en/latest/>

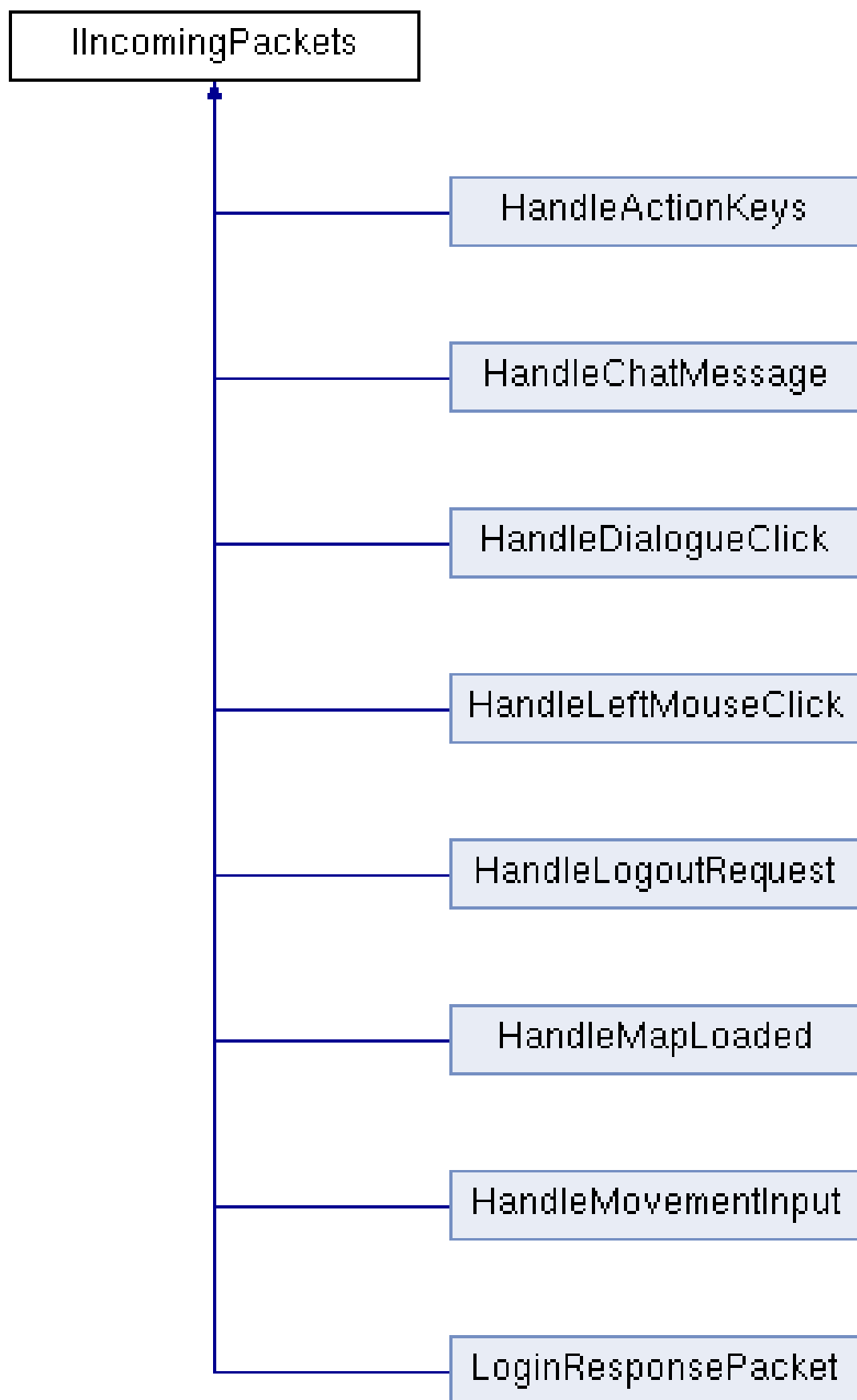
<https://autofac.org/>

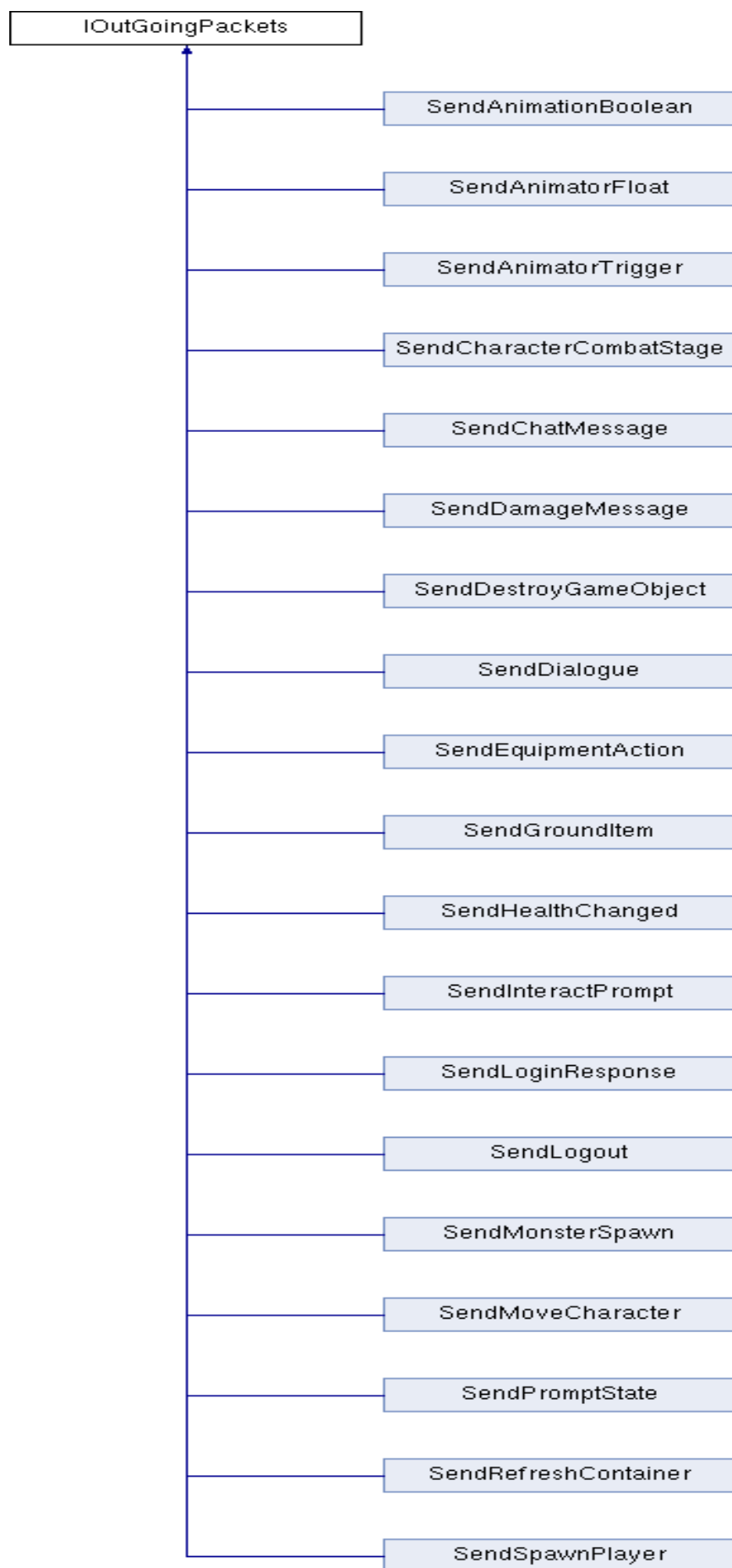
We consider all incoming packets part of a server container along with world classes. The only thing we don't treat as the server container dependencies are the monobehaviour classes that run on the unity main thread

to control game objects and their components. Below are network diagrams. And Inheritance implementation diagrams.









Server Systems And Gameplay

This page consists of all the systems used in the game and a description of the classes used.

Abstract Character Class

Character class is derived by

Player Class

Npc Class

Both classes use the same movement and components, but players have other progress fields to manage like Quests and stats for equipment.

Container System

- A container consists of slot items and management of these slots. We can use this for things like a shop container with shop items to handle appropriately. We only use this implementation for our hotkeys. We are allowed to add slot items, remove them, and refresh the container to apply ui Effects. We can also activate a slot with our hotkey inputs (1-9 on keyboard)

Player Data Save And Load

Health

Hotkey Items

LastActive equip

Equipment System

Simple system to equip weapons and unequip them. We use breath first search to find a descendant child game object of the equipment.

Dialogue System

- Has Multi dialogue transition statements,

- Custom dialogue options for different states a player should be in for the dialogue message

Quest System

- The quest system saves player progress once started
- You can start the quest through the dialogue system
- Reward is given when quest is completed
- Dialogue that gives you a quest changes based on quest progress

Interaction System

- 3 types of interacts
 - Dialogue
 - Ground Item
 - Monster
- Each Interact has unique colliders to trigger their appropriate functions
- On interact enters player set current interact
- When the F key is pressed you perform an interaction on whatever interactive item you have. Depending on the Type of interaction, it will handle your special interaction accordingly. Pickup items, destroy game objects, or prompt dialogues. Or If its a monster it will trigger combat aggression

Combat System


- Setup the default animator with appropriate events
- Had to research a good way to detect collision for melee needs
- Went with tedious route of doing collision boxes on the "Handle" to collide with
 - Handles are currently set to Left and Right Hand
 - Created a listener which gets attached to a Handler
 - Created a CombatAnimation to connect the appropriate animation events
 - CombatAnimation control properly enabled disable the respective handle to listen on
- Additional packet was created for animator "Trigger" parameters to be actives
 - sends name length and the encoded bytes for the string
 - packet reads int and then reads the bytes required for the size of message

- During handle listen, all collision entries with a CombatController will Call an ApplyHit(attacker)
 - ApplyHit now reads the definition properties of what is attacking
 - Damage is calculate on hit and not before the hit right now which I might change for better efficiency
 - Apply hit now triggers our OnHit animation to play, receiving damage on all clients.
 - The collision method wasn't working properly turns out "Box Colliders" don't act indecently meaning i can't have multiple box colliders trigger an event
 - I ended up keeping the box colliders and having the triggers on still since they are smaller than the bigger one it doesn't really trigger what's already triggered...
 - Using this I ignored the defaults events, and basically got all the colliders touching the bounding box during the animation event.
 - Apply the same "ApplyHit" as before and now we have decent hit reactions
 - Did some combat method stagey changes just some basic ones for now
 - includes attack rates (to give gaps per attack)
 - pause on attack (no movement)
 - release time passed after attack to allow to move if we need to
 - WithinReach function to check if I'm within distance to attack

Movement System

- **Our movement system is simple, we use WASD to move and send it to the server**
- **Visual client changes are sent at a server send rate so we do stutter to much**
- **Clients lerp to the desire goal every 200 ms**

Server Classes

ChannelEventHandler	This class is our main network channel class. It is used to register sessions or clients to the server. It will create a PlayerSession class for each socket channel created. This class is an implementation for an asynchronous network so all tasks in here including reading incoming messages are non yielding. Meaning all player sessions created will not halt any time the server receives more messages
ChannelPipeLineHandler	This class implements our Dotnety class. It's a required implementation for the framework channel
Character	
CombatAnimations	Monobehaviour class given to every character type object such as a player or a monster. We control all collider data here as well including times when we have weapons equipped. We keep track of used transforms sort of like a cache just incase we need to use the child transform for a collision detection
CombatComponent	This is a class used for all characters that allowed to perform combat behaviors. This includes Players and Monsters. We keep track of all combat related data including max combos for the animation tree to cycle through. Damage data, and we also have a list of possible targets to filter through when we want
Container	<p>This class is used to manage any group of Slot Items. We only implement this class currently with Hotkeys. Hotkeys A container adds/removes, and refreshes. We also listen for unique slot clicks in the implementation</p> <p>See also</p> <p>Hotkeys.HandleSlotUse(int)</p>
 CustomHeap	<p>This class was made to demonstrate my understanding on what a Heap is and how to properly use them to get optimal sorting algorithm. Although this class is now deprecated. The main purpose was to be able to update heap items based on their respect heap index. We used this for our pathfinding class to add node positions in our list possible nodes to target.</p> <p>PathFinding.FindPath(UnityEngine.Vector3, UnityEngine.Vector3)</p>
Dialogue	This class is used to represent a Dialogue . A dialogue consists of possible message boards or an array of strings. Dialogues have a corresponding array of strings to represent options available for message index Dialogues have a dialogue id and a dialog title as some

DialogueOptions

identifier variable. **DialogueSystem** for implementations

This is an abstract class we use to handle custom dialogue options for any dialogue during runtime

Implementation**DialogueSystem**

This class is a monobehaviour class which gets added as a component to a game object. This dialogue systems It sets up all our default dialogues. Currently we only have 1 dialogue with a quest and reward system applied to different dialogue states

DialougeInteract

This class is used to create a dialogue game object. Anytime you put this mono behavior to a gameobjects component you can set the description. Its important to include a collider as well on the gameobject

Equipment

This class is used to control active weapons and armors equipped to parent gameobject. We trigger the functions through invoke calls from the HandleEquipment Packet

GroundItem

This class is used to create a ground item game object. Anytime you put this mono behavior to a gameobjects component you can set the description. It's important to include a collider as well on the gameobject

HandleActionKeys

This class is used to handle action keys such as all the keyboard numbers (1-9) and escape key

HandleChatMessage

This class will be used mainly to ExecutePacket and read our message buffer data. Then it will attach our senders server username to the message This message will then be sent to all players to display on chat

HandleDialogueClick

This class will handle the option clicked index value to a player's current dialogue option message set. **DialogueOptions**

See also

Player.ActiveDialogue

We execute this on the unity main thread because we want to add items possibly on a dialogue option and that requires access to gameobjects which only exist on unitys main thread

HandleLeftMouseClicked**HandleLogoutRequest**

This class is used to handle the escape screen logout request. The reason we want the server to treat it as a request is because we want to validate if its possible to log the session out

HandleMapLoaded	This HandleMapLoaded packet is executed after our client has logged in successfully and the event is sent from the client to indicate our map scene is done loading all the game objects it needs. Once we know the map is loaded we can spawn all our current players to their client along with ground objects server has spawned, and all the monsters
HandleMovementInput	Anytime a client has sent a move vector we will be using this class to read the move vector values. We execute the gameobject movement on the UnityMainThread using the Dispatcher. UnityMainThreadDispatcher.Instance See also MovementComponent.Move(UnityEngine.Vector3, bool, float)
Hotkeys	This class is used to handle player hotkey slots. It is treated like a Container. We store items in here and we also save and load them during player login. We can trigger the slot as well based on our input controller. See also Container See also InputController.UseHotkey(Player, int)
IConnectionManager	This interface class is used to inherit DotNettys ChannelHandler. ChannelEventHandler
IIncomingPackets	This interface is used to setup all our incoming packet types to be executed appropriately. Check the manual for the tree diagram for inheritance. All packets inherit from this class and call the executable function to read the buffer message in order
IInputControl	This class is used to handle specific input types received from the client. Only the special keys in our enum. KeyInput See also InputController
IItem	Temporary item class for Heap Items
ILoadMapData	This was a starable interface used to build our map container which was used for pathfinding behaviors. We no longer use this because we switch to navmesh pathfinding. MapData

InputController	<p>This is our main input controller class. In here we handle hotkeys and "F" keyboard input for interactions. Interactions for F Key are set by their respective collider trigger functions</p> <p>DialougeInteract.OnTriggerEnter(UnityEngine.Collider)</p> <p>See also</p> <p>MonsterInteract.OnTriggerEnter(UnityEngine.Collider),</p> <p>GroundItem.OnTriggerEnter(UnityEngine.Collider)</p> <p>Hotkeys inputs from 1-9 keyboard keys will trigger a hotkey slot.</p> <p>Hotkeys.HandleSlotUse(int)</p>
IOutGoingPackets	Interface class used to implement all outgoing packets
IPacketHandler	This class is the main packet handler. All incoming packets will go through this class. Each player session is also setup at this point
IPathFinding	Pathfinding interface is deprecated code. We had it as an interface to build it as part of our server dependencies.
IPlayerDataLoader	NetworkManager.RegisterDependencies(Autofac.ContainerBuilder)
IServerTCP	<p>This interface is used as a signal instance dependency for our server container</p> <p>This server interface is used to implement the single instance dependency.</p> <p>NetworkManager.RegisterDependencies(Autofac.ContainerBuilder)</p>
ItemBase	<p>This class is used to attach to any game model. There's no check for what kind of item type because we will be setting and refreshing based on specific values being set. Values are ignored for visual representation on clients. Server just keeps them all as 1. Items can be more than just weapons</p>
IWorld	This interface implements the entire game world. It its in control of what exists in the server and also the initial startup tasks
KeyValuePair	This is a serializable class for unity to display a special key value pair on the inspector. Since unity doesn't show a dictionary on inspector i had to create a special key value class to represent a list item
LoginResponsePacket	<p>This is the packet that gets handled after a player attempts to login. They will open a channel and send then input username and password. This packet will then get executed to validate the information. Depending on the input validation we send a response code to the client. The client handles the response on the login screen with a message prompt if it was not a valid login. If it was a valid login, it continues to load the map and bring them to the game world</p>

MapData	This class is now deprecated but was used to load up the games map data in grid format. We used this grid to create a pathfinding algorithm using A* implementation. IPathFinding implementation. This was a very early implementation of what I wanted to accomplish in this project. Ended up deprecating it
MonsterInteract	This class is used to set up an aggressive monster game object. We set a big collider around the gameobject trigger our collider events OnTriggerEnter and OnTriggerExit
MovementComponent	All character game objects that move and replicate to all clients will have this movement component. It is used to send the movement direction at a fixed network send rate. We have some basic movement data as well. A player is driving this component based on keyboard input while a Server monster is creating its own move vector based on the target information we have. We can use relative position and goal rotations to create our move direction to send to all clients
NetworkBuilder	This is the main network class. We create our Dotnetty classes here, and define our login headers with the required Dotnetty header values to all possible connections. Normally we would expand this and encrypt it with our own login headers making it so no other client versions can connect easily. It's important to realize this is just setting up the global ChannelEventHandler functions. to listen when the server network is notified to do so. Dotnetty is easy like that, takes care of the complicated mess to a multi threaded asynchronous network
NetworkManager	Networkmanager is the heart of the server's game builder and network creation. Its a MonoBehaviour class because that's how unity triggers a startup file The purpose of this class is to build the container with all dependencies the game relies on Dependencies include all our Interfaces
Node	This class is used to represent a node item in our heap
Npc	This class is used to create any server npcs which is of type character. We inherit some base data from characters such as the positions and rotations we use do send over the network But we also have unique function calls that only behave strictly for an NPC type. Along with some game object setup details for any combat state npcs. Currently we only have room for the combat state npc
NpcDefinition	This class is used to define all npc properties. Npcs can either be monsters or interactable npc
PacketHandler	This is the main packet handler instance class. All incoming messages go through this packet filter. It's important to realize that this is a single instance dependency class for the server. This means

PathFinding

any constructors calling **IPacketHandler** are pointing to the instance in memory to use

This class is no longer used, it is considered Deprecated code. We use default nav mesh agent to do any pathfinding

PathResult

Class to Represent a path that is finished

Player

This class is use to create any server players which is of type character. We inherit some base data from character such as the positions and rotations we use do send over the network But we also have unique function calls that only behave strictly for an **Player** type. Along with some game object setup details for any combat state npcs

PlayerData

This class is used for **Player** session saving and loading. We serialize the data and save it as a .json text file. We also deserialize it the same way and initialize the respective player session channel data. This class is also a dependency build which means we can pass it as an instance container in any other Constructors in the container build.

See also

NetworkManager.RegisterDependencies(Autofac.ContainerBuilder)

PlayerSave

This structure is used for serialization deserialization purposes. Json is the form of serialization we use to save a player file. We don't need any gameobject logic that's defined in **Player** class so we just have the struct represent everything we need

PlayerSession

This class is created for every player on login attempts. Even invalid ones. We need this session open to send packets back to a client. We register our global dependencies here too because a player might need to inject them in their memory space.

ChannelEventHandler.ChannelRegistered(DotNetty.Transport.Channel s.IChannelHandlerContext) When i say inject i just mean assign a pointer value to the single instance dependencies.

See also

NetworkManager.RegisterDependencies(Autofac.ContainerBuilder)

Quest

This class is used to keep track of any quest and their progress. We save a serialized version for a player to load and save too. We also create our first quest called "Basic Quest" in **QuestSystem**

QuestSystem

This class is created as a component for the world manager game object. It runs on and creates the default quests available for the game. We use this class to increment current quest steps and give players quests if they don't already have one

SendAnimationBoolean

This class is used to send animator boolean changes to all clients or specific clients. The packet sends a buffer with the server id of the gameobject getting animation change, the name of the animator

SendAnimatorFloat

boolean parameter to change and the value to set the parameter to
 This class is used to send animator float changes to all clients or specific clients. The packet sends a buffer with the server id of the gameobject getting animation change, the name of the animator float parameter to change and the value to set the parameter to

SendAnimatorTrigger

This class is used to "Trigger" Animator parameter values. Trigger is a type of animator field

SendCharacterCombatStage

This class send and packet to the client to alter the animators combat stage so we can visually see what animation we should be using

SendChatMessage

This class is executed to all clients when a player send any chat message. **HandleChatMessage** This class gets created with a message to send to other clients which will then be displayed on their respective chat boxes

SendDamageMessage

This class is created when we want to send a damage indicator to some game object. Mainly just Characters. All characters have a guid to let client map to the proper gameobject to create damage visuals for

SendDestroyGameObject

This class is used to destroy any game object id. We don't use this to destroy server objects we just use it to signal all clients which server object was destroyed so we can destroy it on the clients accordingly

SendDialogue

This class is used to send a dialogue message to a client, We attach the options as well because we let the client show them once the typing is finished Client should already know what options even before its finished

SendEquipmentAction

This class is used to send any kind of equipment action to a client. This includes Equipping and Unequipping. We use this class to send details on the equipment as well, like what parent child should this weapon equip to and the weapon name so the client can load resources if it needs to. WE also send the main server id of the object getting equip because client should know which id is getting the equip

SendGroundItem

This class is responsible for sending a ground object that is being spawned on the server to all clients. Or a specific client, currently we just have any ground object being spawned sent to everyone on the client and everyone that connects afterward also calls this packet

SendHealthChanged

This class is used to replicate a server players health bar to a client causing the green hp bar to scale accordingly. We send the percent covered because we just want to scale the ui based on size and don't really care about visual health values

SendInteractPrompt

Anytime an interaction is triggered on the server, this packet is then sent to the colliding player. It will trigger the prompt panel and change the message

SendLoginResponse	Anytime a player performs a valid login, we send him the server id generated and the position of spawn
SendLogout	This class will just return a player back to a login screen, for other players that are not the person logging out, they will call SendDestroyGameObject packet. SendDestroyGameObject
SendMonsterSpawn	Anytime the server wants to spawn a monster it will send this packet to all clients. World.LoadMonsters for call reference
SendMoveCharacter	This packet gets sent to all clients for ALL character movements. All characters get a Movement Component which then calls a SendMoveCharacterPacket every 200 ms to all clients. MovementComponent.Move(UnityEngine.Vector3, bool, float)
SendPromptState	Anytime we want to trigger a prompt to be visible we call this packet. We use this for dialogues and prompt the logout currently. We also use this class to Close dialogue prompts. It's treated as a toggle prompt. The name you write has to exist on the client as a Tagged game object. Otherwise it won't do anything
SendRefreshContainer	Anytime we want to refresh a container we are going to send this packet. A refresh consists of an array of SlotItems. We are going to send the number of items and constructor the bytes array for what slot items exist in our container. We only use this class for Hotkey Items. Hotkeys.RefreshItems()
SendSpawnPlayer	This is similar to our other spawn game object methods where we attach a server id and position values. Reason we have them all separate is so that clients can handle the list management appropriately. Its also a lot more clear to have a packet for each signal just for clarity sake Anytime a player is being spawned all clients will get this packet being sent to them. After a client loads there map this packet is being sent to register the new client to everyone and to register everyone else to my new login client. HandleMapLoaded.ExecutePacket(Player, DotNetty Buffers.IByteBuffer)
ServerBooter	Main server booter Its called after the network container is built for network NetworkManager
SlotItem	This class is mainly used to keep a collection of items to save and load. Along with use during runtime
UnityMainThreadDispatcher	A thread-safe class which holds a queue with actions to execute on the next Update() method. It can be used to make calls to the main thread for things such as UI Manipulation in Unity. It was developed for use in combination with the Firebase Unity plugin, which uses separate threads for event handling
World	The world class is to control everything going on in the world. We also boot up the server with our 3 tasks defined. We have to do these tasks before we start up a server

Client Framework Design

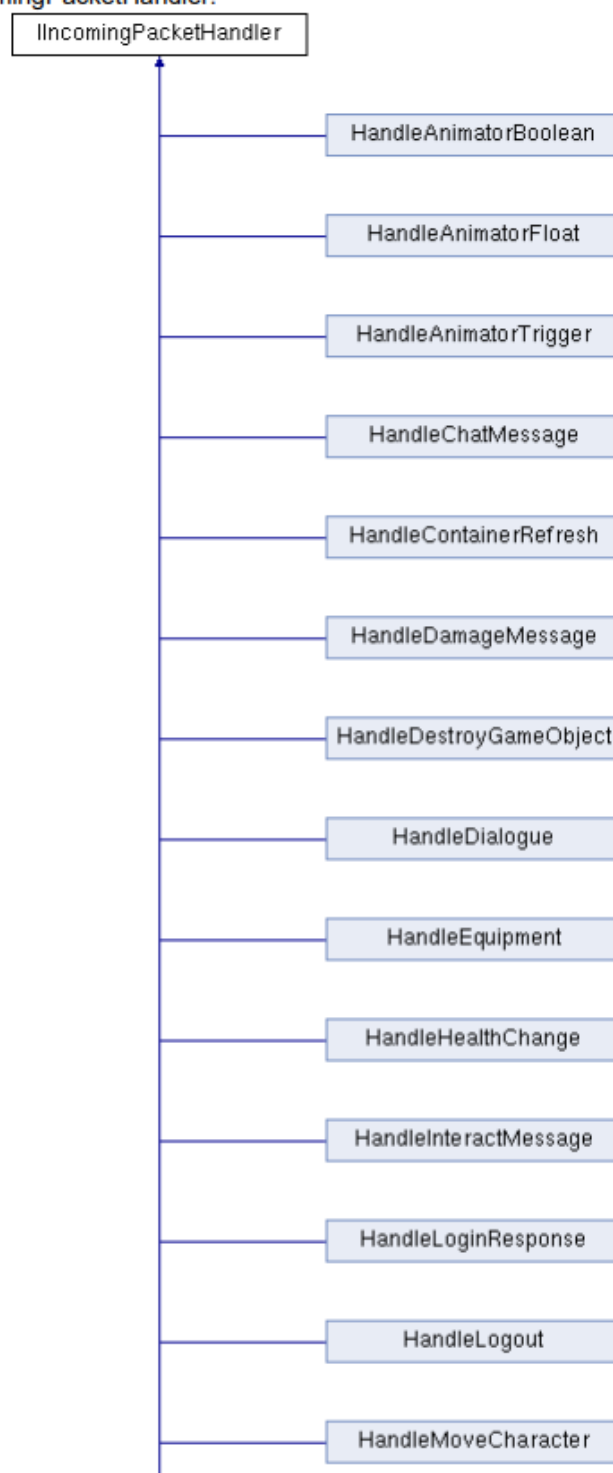
The client uses the DotNetty Library as well to implement our network channels and session handling for data sending.

<https://github.com/Azure/DotNetty>

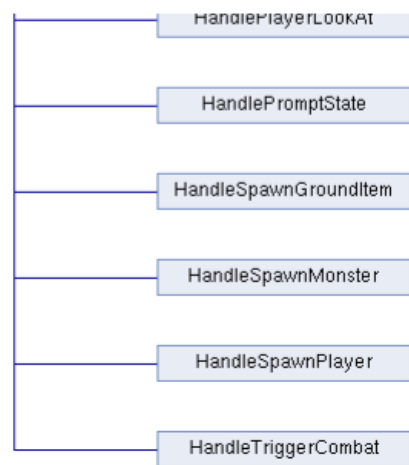
I didn't use autofac on the client because it was just a bigger mess to deal with and it was too late to turn back.

See Below for packet inheritance diagram and Network channel builder diagrams for.

Inheritance diagram for IIncomingPacketHandler:



HandlePlayerLookAt



Client Classes

ChannelBuilder

This class is used to create our main network bootstrap using Netty

ChannelEventHandler

This class is an adapter inherited class of DotNettys ChannelHandlerAdapter This class is used to register the channel signals for a server and client communication. This includes the initial network register, packets available map definitions, The ReadChannelEvent and Unregistering the channel See also ChannelBuilder for instance setup details

▼ ChatManager

This is the main class to handle our chat window. We listen for player input here to send to the server which then sends it back to everyone including myself. Anytime we receive messages we create a new object for its line and change the text accordingly and parent it to our chat window. **ReceiveChatMessage(string, int)**

TabInfo

Serializable class that represents chat tabs. We only use one type of category for now, but can support more.

Container

This class is used to represent anything with slot data and ItemActions. Currently we only add this class to the Hotkey UI gameobjects. Our main packet to control a container is just used to refresh all the items with what the server has each time the slot is updated. The reason we update all slots and not just 1 is because a client can have changed visual representations of another slot but perform the server authoritative update on 1 slot only. So we just do all the slots. Most if not all containers are very small size under 500 so the refresh time is not long.

See also

HandleContainerRefresh

DamageLife

This class is a monobehaviour and is attached to a prefab. We parent the prefab to the workspace area to trigger the start behavior. We use this class to create the visual representation for incoming damage and the lifetime it should stay for. This prefab contains this class gets destroyed after the lifetime is reached resulting in unity cleaning itself up

DialoguePrompt

This class handles any dialogue prompts we receive from the server. We also setup the option buttons here for the server to handle respectively to the dialogue

Equipment

This class is used to control active weapons and armors equipped to parent gameobject. We trigger the functions through invoke calls from the

HandleEquipment Packet

EscapeKeyScreen

This class handles input detection for our escape screen. This displays everything related to the logout screen

GameManager

This is the main class for our game. We use this to spawn game objects/players/and local player requirements says input controls and camera controls. Without this class the game doesn't work

HandleAnimatorBoolean

This packet is used to set any boolean parameter value by name for specific game object id s' Animation Controller

HandleAnimatorFloat

This packet is used to set any float parameter value by name for specific game object id s' Animation Controller

HandleAnimatorTrigger

This packet is used to activate any trigger by name to any gameobject

HandleChatMessage

This class is used to handle incoming messages
Display all messages to our **ChatManager**
ChatManager

HandleContainerRefresh

We use class to handle all our container refreshes.
Containers contain Slots. They are used for Shop

HandleDamageMessage

items, inventory items and hotkey items currently

This class is used to start the life of a damage text prab. A damage text prefab has the **DamageLife** monobehaviour class. **DamageLife**

HandleDestroyGameObject

This class is used to destroy any server game object including other players (not local)

HandleDialogue

This class is used to set up any dialogues that are being prompted to our local player. We read in the the options needed for the buttons too

HandleEquipment

This class is used to perform equipment actions to any gameobject spawned by a server. Equip actions are Equip and Unequip. We execute game object changes on the unity main thread to allow change

HandleHealthChange

Anytime we have a health bar to change we will apply the value changes here

HandleInteractMessage

This class is used to handle incoming messages

Display all messages to our **ChatManager**
ChatManager

HandleLoginResponse

This class is only used to setup a valid login attempt, it will unload our login scene and setup the map scene and spawn the actual local player

HandleLogout

This class destroys a player gameobject during logout (if other players disconnect it will only destroy the game object here)

HandleMoveCharacter	This class is used to handle all character object movements performed on the server. We have to interpolate between its current position and its receiving positions to account for some delay. This would look smoother with some local movement which i don't have yet but i may consider adding
HandlePlayerLookAt	This class is used to trigger any look camera actions by the server. When we set this our camera will and movement information will be relative to the target we are looking at. It will also force you into a strafe mode
HandlePromptState	This class is used to handle incoming messages Display all messages to our ChatManager ChatManager
HandleSpawnGroundItem	This class is used to spawn all ground items. We load all the resources on runtime on the main unity thread in this class
HandleSpawnMonster	This class is used to spawn all monster models. We load all the resources on runtime on the main unity thread in this class
HandleSpawnPlayer	This class is used for all player spawns except for local player. Local player is handled by our loginresponse
HandleTriggerCombat	This class is used to control a game objects animator state for combat. We only use this as a toggle packet meaning its only execute to enable the triggers not

HealthBar

disable because the animators treat "Triggers" like this

This class is mainly used to update the health bar UI on our client. We update it from our network packet.

HandleDamageMessage**IIncomingPacketHandler**

Interface class used to implement all incoming packets

IOutgoingPacketSender

Interface class used to implement all outgoing packets

KeyListener

This class is added to any Local Game object
 GameManager.SpawnPlayer(string, System.Guid, UnityEngine.Vector3, UnityEngine.Quaternion, bool) We listen for movement keys here and send our expected move vector to the server. That's it, we have no local movement currently because we let our server handle how we move and where we move to. We also take into account camera look vector and apply that to the movement vector so our server transforms are relative to our look move vector

LoginScreen

This class is used to handle the login screen portion of the game. Its important to realize when we are on this screen, we also don't have a valid network available. We only create the socket connection after we hit the login button and create a temporary state for the response code received by the server to be handled appropriately. **HandleLoginResponse**

MouseInputUIBlocker

This class is used to block ui elements from being clicked when you are hovering other ui elements. For instance clicking on escape panel while trying to do combat in game will be blocked because of ui being in the way

MoveSync

This class is used to lerp all server movement for any game object. It's not perfect but it kind of works. [HandleMoveCharacter](#) We attach this to all monster or player game objects [HandleSpawnMonster](#) See also [HandleSpawnPlayer](#)

NetworkManager**PlayerCamera**

This class is used to control a Player's Camera. We instantiate a new player camera anytime we are actually logging in. `GameManager.SpawnPlayer(string, System.Guid, UnityEngine.Vector3, UnityEngine.Quaternion, bool)`

SendActionKeys

This class is created each time we are trying to send a `KeyInput` type. The input type is just an enum ordinal which also exists on the server. `KeyInput`

SendChatMessage

This class is used to send any chat message from our ChatManager's input field. We send it to the server, and then get a message back from server containing the actual message with username attached of my player

SendDialogueOption**SendLoginRequest**

This class is created anytime we try to send a login

SendLogoutRequest

request from our login screen. [LoginScreen](#)

This class is used as an empty packet to signal a logout request

SendMapLoaded

This class is used as an empty packet signal to let server know we are done loading the map

SendMouseLeftClick

This class is used to send a signal to server indicating we are using left mouse button

SendMovementPacket

This class is used to send a local player's relative movement vector and any strafe input that is being held. The server uses this to move their server game object which we then receive a packet for.

HandleMoveCharacter**UnityMainThreadDispatcher**

A thread-safe class which holds a queue with actions to execute on the next Update() method. It can be used to make calls to the main thread for things such as UI Manipulation in Unity. It was developed for use in combination with the Firebase Unity plugin, which uses separate threads for event handling

Utils

Project Specifications

Overall: I believe this game will just be about having fun dodging and attacking monsters. The actual gameplay to support this type of atmosphere would not be able to be completed in a fair amount of time.

The Gameplay will consist of

- Replicated movement for everything on the server (models and animations) - we don't play the animations on the server.

- Combat system for player which will also involve replicating to all clients
 - Strafing
 - Dodge key
 - Attack buttons
 - Npcs will try and be intelligent enough to make combat somewhat fun and

challenging. I enjoy combat games, that's why i want to challenge this. Doing this

from scratch is not easy and does cause major setbacks on any full game development. High level combat systems are extremely praised on the indie dev

world, and this would just be a really strong attempt to get something fun to battle

With.

- Camera system to best fit the combat
- There is no solid objective with this game so it's not going to be a life time of fun.
- 1 Npc to talk with to get a quest and a monster npc
- We will have a basic map with some props placed
- A character model which will also be used as a monster model
- Full animations on the character model
 - Strafing (directional movement including diagonal)
 - Default attack animations (with no weapons)
 - Weapon attack animations (based on number of weapons added)

- By default we will just use 1 getting hit reaction animation
- Character will have a weapon or 2, i'm going to try and add 2 (1 sword 1 bow)
- Player movement will be relative to its forward movement by default
- Player movement during combat will turn in relative to camera movement making player rotate with the camera and keep both looking at and moving in the same direction.
- Most importantly we will have all models (including players) being replicated to all clients appropriately. To look really clean and smooth.
- I'll have a 2 huds for a player, a health and hotkey area where u can activate items

Controls:

- We will be using keyboard input
- Movement keys will be WASD
- Mouse will always rotate the camera
- Right mouse button will trigger combat Aim mode, putting you in strafing mode.

Conclusion

The goal of this project was to build a unity game using my own network implementation. I believe I achieved this goal. There were a lot of complications with the project during the initial development, but after enough research I found a way to overcome my problems. As i developed this project i saw i had bad coding practices in some of my system designs. But I was forced to continue because if i had stopped to rework the framework there would be more delay for the submission.

In the project I managed to cover all the core network portions of a multiplayer game. Such as replicating movement, and having no interruptions when communicating with servers and other clients. I do have network lag, but i believe that can be fixed with a better movement method other than Character Controller provided by unity. If i had used a velocity movement method it could look a lot smoother.

I learned what goes into making a game, start to finish. On top of all the networking done behind the game, the game framework runs incredibly smooth in terms of memory cost and cpu usage. We are barely hitting 1% with 3 players running on the server. This is not crazy but it shows the memory management behind a large scale application is there and managed.

This project was a great experience. Although it was stressful and scary, it feels good to complete such a large project with a report this large. I can't picture

my life like this all the time in the work field. Where I'm required to write massive amounts of documentation. I think I have all the tools to find a job somewhere in the game development world. This project will be a big benefit to my github repository and resume.

Libraries and References Used

<https://autofac.org/>

<https://github.com/Azure/DotNetty>

<https://github.com/PimDeWitte/UnityMainThreadDispatcher>

Unity Assets Used

<https://assetstore.unity.com/packages/3d/animations/rpg-character-mechanics-animation-pack-free-65284>

https://assetstore.unity.com/packages/3d/characters/low-poly-survival-characters-174721?aid=1101l8vAv&utm_source=aff (this link was removed now it was up and free during the time of download)