Multiplayer game with Unity and Node.js #2- Connecting with others and Spawning new players

1. Open the server package (in virtual studio) and make a “Classes” folder, and create a file inside called “Player.js”. You will be using “this” function a lot referring to the player inside this constructor.

var shortID = require(‘shortid’);

module.exports = class Player {

constructor() {

this.username = ‘’;

this.id = shortID.generate();

}

}

Now switch back to index.js (that was created in the first tutorial of this series.

Its time to update your index so var Player updates based on this script. After we need to store the players in the server. A dictionary will be used to map the players easier. In the server we create a player after a connection was made.

var io = require(‘socket.io’)(process\_env\_PORT || 52300);

//Custom Classes

var Player = require(‘./Classes/Player.js’);

console.log(‘Server has started’);

var players = [];

var sockets = [];

io.on(‘connection’, function(socket) {

console.log(‘Connection made!’);

var player = new Player();

var thisPlayerID = player.id;

players[thisPlayerID] = player;

sockets[thisPlayerID] = sockets;

//Tell the client that this is our id for the server

socket.emit(‘register’, (id: thisPlayerID));

socket.emit(‘spawn’, player); //Tell myself I have spawned

socket.broadcast.emit(‘spawn’, player) // Tell others I have spawned

//Tell myself about everyone else in the game

for(var playerID in players) {

if(playerID != thisPlayerID) {

socket.emit(‘spawn’, players[playerID]);

}

}

socket.on(‘disconnect’, function() {

console.log(‘A player has disconnected’);

delete players[thisPlayerID];

delete sockets[thisPlayerID];

socket.broadcast.emit(‘disconnected’, player);

});

});

on function is for client to server, emit is from server to client. Broadcast sends it to any other socket connected, except yours. Now turn on the server via a terminal in visual studio

nodemon index.js

1. Now time to update unity. Create an empty object in Unity and class it EmptySpawnObject.

We must also update the NetworkClient script. This is for spawning players, the player id is recognized for data passing through, and we are going to create and map the game objects to our dictionary. We use the special functions register spawn and this is great to incorporate with mysql databases for exchanging server leaderboards etc. Remember to do the script initialize after a login page, which can be easily created.

Add this network client script to the empty game object.

using System.Collections

using System.Collections.Generic;

using UnityEngine;

using SocketIO;

namespace Project.Networking{

public class NetworkClient : SocketIOComponent{

[Header(“Network Client”)]

[SerializedField]

private Transform networkContainer;

private Dictionary<string, GameObject> serverObjects;

public override void Start(){

base.Start();

initialize();

setupEvents();

}

public override void Update(){

base.Update();

}

private void initialize() {

serverObjects = new Dictionary<string, GameObject>();

}

private void setupEvents() {

On(“open”, (E) -> {

Debug.Log(“Connection made to the server”);

});

On(“register”, (E) -> {

string id = E.data[“id”].ToString().RemoveQuotes();

Debug.LogFormat(“Our Client’s ID ({0})”, id);

});

On(“spawn”, (E) -> {

//handling all spawning all players

//passed data

string id = E.data[“id”].ToString().RemoveQuotes();

GameObject go = new GameObject(“Server ID: “ + id);

go.transform.SetParent(networkContainer);

serverObjects.Add(id, go);

});

On(“disconnected”, (E) -> {

string id = E.data[“id”].ToString().RemoveQuotes();

GameObject go = serverObjects[id];

Destroy(go);

serverObjects.Remove(id);

});

}

}

Now time to add that empty game object to the code- networking empty object in the newly created variable for containing player ids. This is just the beginning to working with both unity and node.js.

Now refer to the unity rpg studies for referring to spawning registering and playing data by creating players and spawn points.