Kristupas Stalmokas

CS3IP E-LOGBOOK PROGRESS

CONSIDERATIONS – HOW I MET THEM

How potential players will find each other online and agree to start a game;

Lobby System where players join a lobby together via code or lobby search and then can start the game with those players.

How to share game state while minimising network traffic and the potential for cheating;

Netcode for Game Objects – objects such as players and game entities / variables which determine the state are shared.

Network traffic is minimized by ensuring the state is only shared when required (when variables change enough like player positions over a certain float value.

Potential for cheating is restricted as the network model is client-server in which the server is authoratative and holds the true state of the game that is shared with clients. Clients have their own state that is similar to the server's as they receive the updates from the server, but local movement and physics are unique to each client for a short time until the client sends the next state update – clients are not able to cheat!

How to ensure synchronisation between players' views of the game world.

Sychronisation is ensured via reconciliation – the local state of every client is compared to the true state of the servers. If the client's states' are different enough from the server's state, then the server's state is shared with the client, synchronising the players to the actual state of the game, otherwise their local state is close enough to the actual state and therefore the game carries on until the next reconciliation call. This call occurs often and always compares the states first.

What to do if the network connection is unreliable or intermittent;

Constantly check the client's RTT and if it is unreliable then display a warning message. The reconcilion method used when player movements, velocity and physics are shared between the client/server will have altered parameters to handle higher delayed clients, allowing for their updated state to be shown more smoothly rather when they have a worse connection to the server. The quality of the game's resolution will also be lowered when the connection is delayed.

Project Plan

Project stage	START DATE: 17/10/2024 (PID started 12/10/2024) Project Weeks												
	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
1.1. Research Network Fundamentals													
1.2. Research Unity Tools/Services													
2 Analysis/Design													
2.1. Design Simple Multiplayer Network													
2.2. Design Simple Multiplayer Game													
3 Develop prototype													
3.1. Develop Network													
3.2. Develop Game													
3.3. Implement Network into Game													
3.4. (O) Expand Implementation													
4 Testing/evaluation/validation													
4.1. Test and Debug Implementation													
4.2. Work Demonstration and Evaluation													
5 Assessments													
5.1. Produce Project Poster													
5.2. Produce and Submit Final Report													
5.3. Produce and Showcase Final Presentation													

MEETING 1 29/10

PROGRESS MADE

RESEARCH FOR LITERATURE REVIEW:

- Researched network fundamentals.
- Researched unity tools/services.
- Researched various network protocols.

NEXT WEEK TO DO

Start initial prototype development

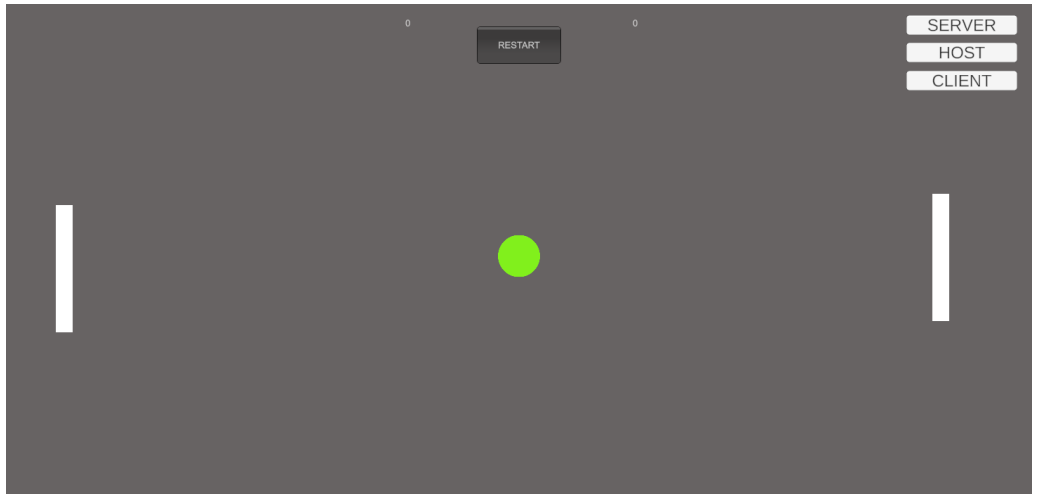
- Install Unity Tools
- Setup Workspace
- Begin development

MEETING 2 12/11

PROGRESS MADE

- Developed simple pong game with UI.
- Uploaded unity project files to GitLab so I can easily work on the project on both my laptop and computer.
- Downloaded and installed essential unity packages for multiplayer functionality – Unity Network Tools and Unity Netcode.
- Began development of simple multiplayer functionality to allow a host and client to connect and play as each paddle.

SIMPLE PONG



Two controllable paddles and a ball with collision that bounces off the paddles and top and bottom walls. With scoring system and UI to display the score, a functional restart button and outputting which player wins when they reach a score value of 10.

Player control

```
0 references
public class PlayerControl : MonoBehaviour
{
    1 reference
    public KeyCode moveUp = KeyCode.W;
    1 reference
    public KeyCode moveDown = KeyCode.S;
    2 references
    public float speed = 10.0f;
    4 references
    public float boundY = 2.25f;
    3 references
    private Rigidbody2D rb2d;


    // Start is called once before the first exec
    0 references
    void Start()
    {
        rb2d = GetComponent<Rigidbody2D>();

    }

    // Update is called once per frame
    0 references
    void Update()
    {
        var vel = rb2d.linearVelocity;
        if (Input.GetKey(moveUp)) {
            vel.y = speed;
        }
        else if (Input.GetKey(moveDown)) {
            vel.y = -speed;
        }
        else {
            vel.y = 0;
        }
        rb2d.linearVelocity = vel;

        var pos = transform.position;
        if (pos.y > boundY) {
            pos.y = boundY;
        }
        else if (pos.y < -boundY) {
            pos.y = -boundY;
        }
        transform.position = pos;
    }
}
```

Wall functionality

```
0 references
public class SideWalls : MonoBehaviour
{

    0 references
    void OnTriggerEnter2D (Collider2D hitInfo) {
        if (hitInfo.name == "Ball") {
            string wallName = transform.name;
            GameManager.Score(wallName);
            hitInfo.gameObject.SendMessage("RestartGame", 1.0f, SendMessageOptions.RequireReceiver);
        }
    }
```

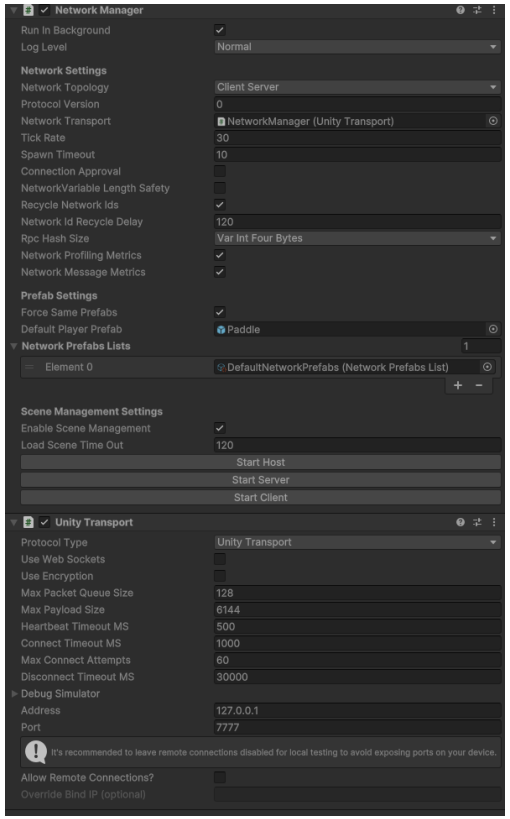
Ball functionality

```
0 references
void GoBall() {
    float rand = Random.Range(0, 2);
    if (rand < 1) {
        rb2d.AddForce(new Vector2(20, ballSpeed));
    }
    else {
        rb2d.AddForce(new Vector2(-20, ballSpeed));
    }
}

// Start is called once before the first execution of Update after the MonoBehaviour is created
0 references
void Start()
{
    rb2d = GetComponent<Rigidbody2D>();
    Invoke("GoBall", 2);
}

1 reference
void ResetBall() {
    rb2d.linearVelocity = Vector2.zero;
    transform.position = Vector2.zero;
}

0 references
void RestartGame() {
    ResetBall();
    Invoke("GoBall", 1);
}

0 references
void OnCollisionEnter2D (Collision2D coll) {
    if (coll.collider.CompareTag("Player")) {
        Vector2 vel;
        vel.x = rb2d.linearVelocity.x;
        vel.y = (rb2d.linearVelocity.y / 2.0f) + (coll.collider.attachedRigidbody.linearVelocity.y / 3.0f);
        rb2d.linearVelocity = vel;
    }
}
```

UI script

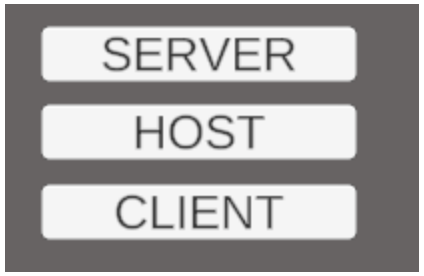
```
1 reference
public static void Score (string WallID){
    if (WallID == "RightWall") {
        PlayerScore1++;
    } else
    {
        PlayerScore2++;
    }
}

0 references
void OnGUI() {
    GUI.skin = layout;
    GUI.Label(new Rect(Screen.width / 2 - 150 - 12, 20, 100, 100), "" + PlayerScore1);
    GUI.Label(new Rect(Screen.width / 2 + 150 + 12, 20, 100, 100), "" + PlayerScore2);

    if (GUI.Button(new Rect(Screen.width /2 - 60, 35, 120, 53), "RESTART"))
    {
        PlayerScore1 = 0;
        PlayerScore2 = 0;
        theBall.SendMessage("RestartGame", 0.5f, SendMessageOptions.RequireReceiver);
    }

    if (PlayerScore1 == 10)
    {
        GUI.Label(new Rect(Screen.width / 2 - 150, 200, 2000, 1000), "PLAYER ONE WINS");
        theBall.SendMessage("ResetBall", null, SendMessageOptions.RequireReceiver);
    } else if (PlayerScore2 == 10)
    {
        GUI.Label(new Rect(Screen.width / 2 - 150, 200, 2000, 1000), "PLAYER TWO WINS");
        theBall.SendMessage("ResetBall", null, SendMessageOptions.RequireReceiver);
    }
```

MULTIPLAYER FUNCTIONALITY



Using unity packages to get this Network Manager script and Unity Transport protocol script.

Basic network buttons




```
0 references
public class NetworkManagerUI : MonoBehaviour
{
    1 reference
    [SerializeField] private Button ServerButton;
    1 reference
    [SerializeField] private Button HostButton;
    1 reference
    [SerializeField] private Button ClientButton;


    0 references
    private void Awake() {
        ServerButton.onClick.AddListener(() => {
            NetworkManager.singleton.StartServer();
        });
        HostButton.onClick.AddListener(() => {
            NetworkManager.singleton.StartHost();
        });
        ClientButton.onClick.AddListener(() => {
            NetworkManager.singleton.StartClient();
        });
    }
}
```

NEXT WEEK TO DO

- Search for online courses on multiplayer network game to start working through or may work through.
- Research example multiplayer games to get an idea of what multiplayer game to make that is more in depth than the pong game – more objects and functions for the network to sync and keep track of which leads to a more detailed network system.
- Figure out what game to implement – either edit the existing pong game to allow for more than 2 players as this will be essential or change it entirely.
- Share the PID, E-Logbook and presentations on progress.

MEETING 3 19/11

PROGRESS MADE

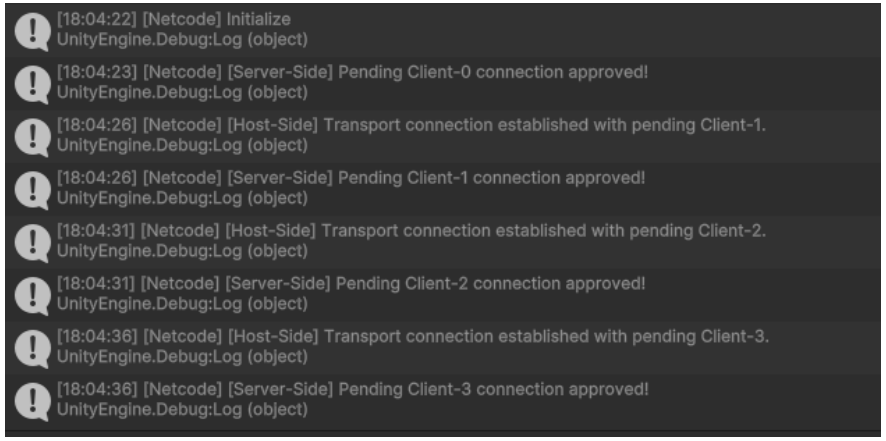
- Found and started multiplayer unity game course.
- Fixed the multiplayer aspect of the pong game and it is now functional.
- Implemented some more data to send through the network.

THE COURSE

https://www.youtube.com/watch?v=3yuBOB3VrCk

This is a short course covering the multiplayer in Unity. So far, I am halfway through this specific video, but the entire course has around 6 videos focusing on different aspects of multiplayer (netcode, authentication, lobbies etc).

MULTIPLAYER PONG



Added spawn locations for paddles – so far a simple case statement on switching spawn locations depending on owner of paddle which allows for 4 total players but can theoretically allow for more if game supports it.

```
void Start()
{
    // Get Player Rigidbody
    rb2d = GetComponent<Rigidbody2D>();

    spawnLocationsX = new Vector2[4];
    spawnLocationsX[0] = new Vector2 (-5.55f, 0.00f);
    spawnLocationsX[1] = new Vector2 (-5.55f, 1.00f);
    spawnLocationsX[2] = new Vector2 (5.55f, 0.00f);
    spawnLocationsX[3] = new Vector2 (5.55f, 1.00f);

    switch (OwnerClientId) {
        case 0:
        transform.position = spawnLocationsX[0];
        break;
        case 1:
        transform.position = spawnLocationsX[1];
        break;
        case 2:
        transform.position = spawnLocationsX[2];
        break;
        case 3:
        transform.position = spawnLocationsX[3];
        break;
    }

}
```

MORE DATA

Implemented sharing of an integer between the network as well as sharing the value when the value changes:

```
// Test for syncing variable value, random integer:
3 references
private NetworkVariable<int> randomNumber = new NetworkVariable<int>(1, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Owner);
// New NetworkVariable int, (starting value, read permissions, write permissions)
// Everyone is able to read the value, only owners of this specific network variable (every player has one) can alter the value of their owned variable
```

```
// Altering OnNetworkSpawn method to see values of variables and data structs:
0 references
public override void OnNetworkSpawn() {
    // When randomNumber value changes,
    randomNumber.OnValueChanged += (int previousValue, int newValue) => {
    // Output OnwerClientID + that owner's value of the randomNumber
    Debug.Log(OwnerClientId + "; randomNumber: " + randomNumber.Value);
    };
    // This is better than using Debug.Log in the update method as we only need to if the value of randomNumber changes correctly between players when it actually does change value
    // so this way there are less repeated messages.
```

```
    // Press 'I' to generate new random number so can test sync between host and client:
    if (Input.GetKeyDown(KeyCode.I)) {
        randomNumber.Value = Random.Range(0, 100);
    }
```

MORE DATA

Implemented sharing of a custom data structure that contains multiple variables and even a string – this required serialization of the new data structure for it to be able to be shared across the network.

```
// New custom data struct:
2 references
private NetworkVariable<MyCustomData> customIntBool = new NetworkVariable<MyCustomData>(
    new MyCustomData {
        _int = 56,
        _bool = true,
        message = "parkinglotscarecrow",
    }, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Owner);
```

```
// Custom Data struct version:
customIntBool.OnValueChanged += (MyCustomData previousValue, MyCustomData newValue) => {
// Output OnwerClientID + that owner's values of the custom data struct
Debug.Log(OwnerClientId + "; randomNumber: " + newValue._int + "; " + newValue._bool + "; " + newValue.message);
};
```

```
// Data serialization (which is handled for set data types like the int in random num) is the process of converting
// data objects in a data structure into a byte stream for storage, transfer and distribution. This is used in sending data
// over objects.
// For MyCustomData, the data within must be serialized:
6 references
public struct MyCustomData : INetworkSerializable {
    4 references
    public int _int;
    4 references
    public bool _bool;
    4 references
    public string message;

    // Importing base serializer:
    0 references
    public void NetworkSerialize<T>(BufferSerializer<T> serializer) where T : IReaderWriter
    {
        // Altering the serialize method for MyCustomData:
        serializer.SerializeValue(ref _int);
        serializer.SerializeValue(ref _bool);
        serializer.SerializeValue(ref message);
    }
}
```

```
[18:13:30] 1; randomNumber: 66
UnityEngine.Debug:Log (object)
[18:13:31] 1; randomNumber: 10; False; MESSAGE REDACTED
UnityEngine.Debug:Log (object)
[18:13:36] 0; randomNumber: 10; False; MESSAGE REDACTED
UnityEngine.Debug:Log (object)
[18:13:38] 0; randomNumber: 53
UnityEngine.Debug:Log (object)
[18:13:44] 2; randomNumber: 33
UnityEngine.Debug:Log (object)
```

```
// Press 'C' to set value of MyCustomData to specified variable values:
if (Input.GetKeyDown(KeyCode.C)) {
    customIntBool.Value = new MyCustomData {
        _int = 10,
        _bool = false,
        message = "MESSAGE REDACTED",
    };
```

NEXT WEEK TO DO

- Figure out what game to implement
- Fix physics error with paddles – player paddles' physics can interact with each other which is currently not intended.

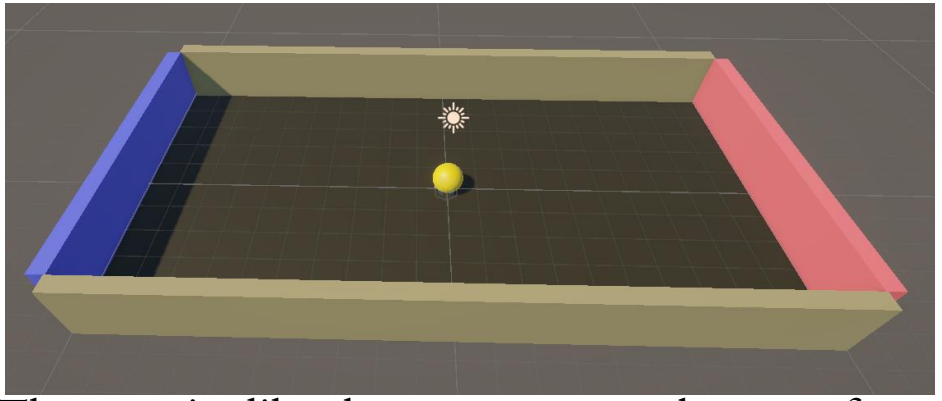
MEETING 4 26/11

PROGRESS MADE

- Developed new whole new game in 3D from scratch.
- Reimplemented the existing network implementation with some editing of code.
- Followed along video course slightly.

NEW MULTIPLAYER GAME

I created a new game in 3D rather than the 2D pong game to allow for more elements – a whole new Z axis to synchronize in both position and rotation.



The premise like the pong game – there are four walls and a ball in which the ball hitting either the left or right wall gives points to the opposing team. The players are balls instead of paddles for now – may expand to allow players to choose whether to be slow wide paddle or a fast small ball.

Currently have opposite issue as last week – players do not collide but also do not run the physics on each other as the client can also not bounce the main ball.

I still must implement the scoring on the left and right wall.

NEXT WEEK TO DO

- Fix physics of client
- Implement full game rules

MEETING 5 03/12

PROGRESS MADE

- Improved movement system (switched to using AddForce to the Rigidbody component of the player rather than changing the transform.position axis).
- Imported Unity Multiplayer Tools (Network Profiler and Runtime Network Statistics Monitor).
- Learnt about how connections between devices are/can be made.

NEW MOVEMENT SYSTEM

MEETING 6 12/12

PROGRESS MADE

- Moved all the movement and physics handling to the server instead of individual players running their own movement scripts. Server has more security and control over clients

MEETING 7 13/02

PROGRESS MADE

- Added local client prediction which removed the 'ghosting' effect which clients see when moving and decreases input latency.
- Added basic scoring system, two goals and user interface for score.
- Added spawn points.
- Started implementation of lobby system

CLIENT PREDICTION

Every client other than host calculate their own movements locally.

They then request for the actual movements from the server via server remote procedure calls

```
0 references
private void FixedUpdate()
{
    if (!IsOwner) return;

    // Apply movement locally first for responsive input
    if (InputKey != Vector3.zero)
    {
        // Local prediction
        Vector3 force = InputKey * acceleration;
        rb.AddForce(force, ForceMode.Force);

        if (rb.linearVelocity.magnitude > maxSpeed)
        {
            rb.linearVelocity = rb.linearVelocity.normalized * maxSpeed;
        }

        // Then send to server
        RequestAccelerateServerRpc(InputKey);
    }
    else
    {
        // Local prediction for deceleration
        if (rb.linearVelocity.magnitude > 0.1f)
        {
            rb.AddForce(rb.linearVelocity * -deceleration, ForceMode.Force);
        }

        // Then send to server
        RequestDecelerateServerRpc();
    }
}
```

```
// Asking for acceleration in movement
[ServerRpc]
1 reference
private void RequestAccelerateServerRpc(Vector3 InputKey, ServerRpcParams rpcParams = default)
{

    // server is authoritative
    Vector3 force = InputKey * acceleration;
    rb.AddForce(force, ForceMode.Force);

    if (rb.linearVelocity.magnitude > maxSpeed)
    {
        rb.linearVelocity = rb.linearVelocity.normalized * maxSpeed;
    }

    // Sync only to other clients, not back to the owner
    var clientRpcParams = new ClientRpcParams
    {
        Send = new ClientRpcSendParams
        {
            TargetClientIds = new ulong[] { OwnerClientId }
        }
    };

    SyncMovementClientRpc(transform.position, clientRpcParams);
    SyncVelocityClientRpc(rb.linearVelocity, clientRpcParams);
}

// Asking for deceleration in movement
[ServerRpc]
1 reference
private void RequestDecelerateServerRpc(ServerRpcParams rpcParams = default)
{
    // Only apply deceleration if we're not in a collision
    if (rb.linearVelocity.magnitude > 0.1f)  // Small threshold to prevent jitter
    {
        rb.AddForce(rb.linearVelocity * -deceleration, ForceMode.Force);
    }

    SyncMovementClientRpc(transform.position);
    SyncVelocityClientRpc(rb.linearVelocity);
}


// Client Rpc
```

BASIC SCORING

```
// Score Handling
2 references
public void Score(bool isPlayer1Scored)
{
    if (!IsServer) return; // Only server can update score
    Debug.Log("Score called for " + (isPlayer1Scored ? "Player 1" : "Player 2"));

    if (isPlayer1Scored) // if player 1 scored
    {
        playerScore1.Value += 1; // update player 1 score
        Debug.Log("Player 1 score is now: " + playerScore1.Value);
    }
    else // else player 2 scored
    {
        playerScore2.Value += 1; // update player 2 score
        Debug.Log("Player 2 score is now: " + playerScore2.Value);
    }

    // reset ball pos
    ResetBallServerRpc();
    ResetPlayerServerRpc();
}

// Getting player scores
1 reference
public int GetPlayer1Score()
{
    return playerScore1.Value;
}

1 reference
public int GetPlayer2Score()
{
    return playerScore2.Value;
}
```

SPAWN POINTS (FOR 2 PLAYERS FOR NOW)

```
0 references
void Start()
{
    // find the ball Game Object
    theBall = GameObject.FindGameObjectWithTag("Ball");

    playerSpawnPos.Add(new Vector3(-5, 1, 0)); // spawn pos 1
    playerSpawnPos.Add(new Vector3(5, 1, 0)); // spawn pos 2
    Debug.Log(playerSpawnPos.Count); // Output num of spawn pos for testing

    // find all players
    GameObject[] players = GameObject.FindGameObjectsWithTag("Player");
    Debug.Log($"Found {players.Length} players");

    // add all players to list of Player Objects
    foreach (GameObject player in players)
    {
        playerObjects.Add(player);
        Debug.Log($"Added player with NetworkObjectId: {player.GetComponent<NetworkObject>().NetworkObjectId}");
    }
}
```

LOBBY SYSTEM

- Using Unity's lobby service, players can host new lobbies and view or join existing lobbies.
- For this to work each player has their own player id via the authentication service provided by Unity.

```
// Start is called before first frame update
0 references
private async void Start()
{
    // Subscribe to the OnClientConnectedCallback event
    // += adds this method to the list of methods to call when this event occurs
    //In this case, when a client connects (including the host), HandleClientConnected will be called
    // NetworkManager.Singleton.OnClientConnectedCallback += HandleClientConnected;

    await UnityServices.InitializeAsync();

    AuthenticationService.Instance.SignedIn += () =>
    {
        Debug.Log("Signed in" + AuthenticationService.Instance.PlayerId);
    };
    await AuthenticationService.Instance.SignInAnonymouslyAsync();

    playerName = "kris" + UnityEngine.Random.Range(10, 99);
    Debug.Log(playerName);
}
```

- The lobby service deletes lobbies that are inactive for 30 seconds by default, so implemented a 'heartbeat' function to keep the lobby alive

```
1 reference
private async void HandleLobbyHeartbeat() // Heartbeat function to keep server alive - by default the lobby service automatically shuts down a lobby for 30 seconds of inactivity
{
    if (hostLobby != null)
    {
        heartbeatTimer -= Time.deltaTime;
        if (heartbeatTimer < 0f)
        {
            float heartbeatTimerMax = 15;
            heartbeatTimer  = heartbeatTimerMax;

            await LobbyService.Instance.SendHeartbeatPingAsync(hostLobby.Id);
        }
    }
}
```

LOBBY SYSTEM

- Creating lobby

```
1 reference
private async void CreateLobby() // Creating the lobby
{
    try {
        string lobbyName = "My Lobby";

        int maxPlayers = 4;
        CreateLobbyOptions createLobbyOptions = new CreateLobbyOptions
        {
            IsPrivate = false,
            Player = GetPlayer()
        };

        Lobby lobby = await LobbyService.Instance.CreateLobbyAsync(lobbyName, maxPlayers, createLobbyOptions);
        lobbyNameUI.text = lobby.Name;

        hostLobby = lobby;
        PrintPlayers(hostLobby);

        Debug.Log("Created lobby! " + lobby.Name + " " + lobby.MaxPlayers + " " + lobby.Id + " " + lobby.LobbyCode);
    } catch (LobbyServiceException e)
    {
        Debug.Log(e);
    }
}
```

- Listing lobbies

```
1 reference
private async void ListLobbies() // Viewing lobby
{
    try {
        QueryLobbiesOptions queryLobbiesOptions = new QueryLobbiesOptions
        {
            Count = 25,
            Filters = new List<QueryFilter>
            {
                new QueryFilter(QueryFilter.FieldOptions.AvailableSlots, "0", QueryFilter.OpOptions.GT) // query filter to see lobbies will available slots > 0
            },
            Order = new List<QueryOrder>
            {
                new QueryOrder(false, QueryOrder.FieldOptions.Created)
            }
        };

        QueryResponse queryResponse = await LobbyService.Instance.QueryLobbiesAsync(queryLobbiesOptions);

        Debug.Log("Lobbies found :" + queryResponse.Results.Count);
        foreach (Lobby lobby in queryResponse.Results)
            {
                Debug.Log(lobby.Name + " " + lobby.MaxPlayers);
            }
    } catch (LobbyServiceException e)
    {
        Debug.Log(e);
    }
}
```

LOBBY SYSTEM

- Joining lobbies via code

```
0 references
private async void JoinLobbyByCode(string lobbyCode) // Joining lobby via code input
{
    try {
        JoinLobbyByCodeOptions joinLobbyByCodeOptions = new JoinLobbyByCodeOptions {
            Player = GetPlayer()
        };

        // QueryResponse queryResponse = await LobbyService.Instance.QueryLobbiesAsync();
        Lobby joinedLobby = await LobbyService.Instance.JoinLobbyByCodeAsync(lobbyCode, joinLobbyByCodeOptions);

        Debug.Log ("Joined lobby with code" + lobbyCode);

        PrintPlayers(joinedLobby);

    } catch (LobbyServiceException e)
    {
        Debug.Log(e);
    }
}
```

- Or joining via quickjoin based on filters set

```
0 references
private async void JoinLobbyByQuickJoin() // Joining lobby via quick join and the filters set
{
    try {
        await LobbyService.Instance.QuickJoinLobbyAsync();

    } catch (LobbyServiceException e)
    {
        Debug.Log(e);
    }
}
```

NEXT WEEK TO DO

- Finish implementation of the basic lobby system
- Ensure spawn points work with the player count

MEETING 8 20/02

PROGRESS MADE

- Further implementation of lobby system
- Fixed scene and script structure – one scene for lobby, one scene for the game

NEXT WEEK TO DO

- What to do if the network connection is unreliable or intermittent
- Finish implementation of the basic lobby system
- Ensure spawn points work with the player count

MEETING 9 27/02

PROGRESS MADE

- Finished basic setup and functionality of lobby system. Players can create lobbies, join other lobbies through a code and leave lobbies. Players get to choose their own name.
- Cleaned up code structure (separated scripts into scripts for handling UI and handling functions).
- Added a display for the latency of players via a ping function which gets the round-trip time from the client to the host. If client has >100ms ping, a warning message appears.

NEXT WEEK TO DO

Finalize the functional prototype demonstration!

- Show a list of players in the lobby (just simple name and update if anyone joins or leaves)
- Add a way for players to reconnect if they lose connection
- Ensure spawn points work with the player count
- Add UI elements
- Toggle runtime network monitor
- More game functions

MEETING 10 06/03

NEXT WEEK TO DO

- Implement network pressure test – choose one, more data to share (random numbers/variable sharing, more players, more advanced functions for players.
- Techniques to solve the network issues / delays to make the system run more smoothly,
- Reconnect to game.

MEETING 11 13/03

PROGRESS MADE

- Implemented network pressure test – more data to share (random numbers/variable sharing
- Implemented minor reconciliation
- Started the Reconnect to game implementation

MEETING 12 20/03

PROGRESS MADE

- Added more data to be shared in the network stress test – a large matrix of size 10000, with every field in the matrix being a random float number that is generated, as well as a size 5000 matrix being shared 200 times per second for every client.
- Fixed 'ghosting' effect that the host's player model appears to have from the client's screen. The only notable issue right now is the clients move slower compared to the host, and when the host collides with something it bounces off, on the clients screen their player model does not hit the wall and just bounces off nothing.
- Added function and UI to search for lobbies that are public and have open space available and choose to join from this search instead of by code.

NEXT WEEK TO DO

- Look at lowering resolution/quality of game when high delay in connection
- Finalize UI implementation of Lobby Search and make sure client joins game
- Continue the Reconnect to game function
- Improve UI design

- Fix Clients Visuals of Host's physics?
- Change network test matrix to be initialized before sharing it?

MEETING 13 03/04

PROGRESS MADE

- Lower resolution depending on the client's RTT (ping) - over 100ms is low resolution, under 100ms but over 50ms is medium resolution, else run at normal resolution.
- Made a new script 'CoreManager' to manage all script instances, including initialization, finding and debugging null instances, enforcing the singleton pattern all the scripts in my game have. This reduces redundancy in my code and is overall more flexible, expandable and cleaner to manage the instances of scripts that every script requires. Whilst this does technically minimize network traffic unnoticeablely, this would mainly ensure that the synchronization between clients and the server is maintained as singleton patterns are maintained for every script (so variables wont be different between players due to different players calling different instances of the same script).
- Made sure clients see lobby information and load the game when joining through lobby search.
- Initialized large matrix for network stress test before sharing. This gives a more realistic stress test as there is barely any cpu/gpu overhead due to initializing such a large matrix so many times, and reduces the packet loss rate if the matrix is shared before being fully initialized. Also added the network profiler to view performance of the game.
- Added username Text in main menu
- Added network IP address and port functionality when joining / hosting games (host IP is saved when starting a lobby via the lobby service, then the client joins through the lobby service and they receive the host IP address and connect via networkmanager.

NEXT WEEK TO DO

- Continue the Reconnect to game function
- Improve UI design
- Fix Clients Visuals of Host's physics, further improve netcode.

MEETING 14 24/03

PROGRESS MADE

- Implemented Unity Relay Service and its functionality within the network. Hosts now create a relay connection where clients connect through to bypass NAT and firewall restrictions between devices. Before, hosts would host through their own local IP address for testing on the same device and moving to a connection on their actual IP address for between devices did not work as the host would not listen for connections. Moving to Unity Relay also allows for devices to connect from anywhere – not only within the same network.
- Added 'ready' functionality in the lobby. All players must agree to start a game.
- Added ping displays in the lobby and in-game so poor connections can be detected early on and during gameplay. (Maybe fully add ability for host to kick players from the lobby)
- Started implementation on game reconnection and saving the game state.

NEXT WEEK TO DO

- Polish final implementations
- Begin Report!