12 - Multiplayer Games and Networking

CS 3160 - Game Programming
Max Gilson

Many Games are Multiplayer

- Old games were multiplayer
- New games are multiplayer
- Multiplayer games are becoming more and more popular
  - Live service games
- Multiplayer games allow the interactions between players to create the experience: emergence
  - Less work for the game programmer (maybe)
  - BUT: Without multiple players, the game is non-existent

Types of Multiplayer Games

- Local (Offline) Multiplayer
  - Split Screen
  - Shared Screen
  - Local Network
- Online Multiplayer
  - P2P
  - Dedicated Server
  - Database
- Hybrid Local + Online
  - Split Screen + Online




Local Multiplayer

- Local multiplayer allows players to play physically next to each other on
  - Usually on the same screen
- Local network (LAN) - players use multiple systems to play on the same network

Local Multiplayer (cont.)

- Local multiplayer means:
  - No internet required
  - Only 1 platform/console/computer required (unless using LAN)
  - More social (couch co-op)
  - No random people playing with you
- Although, you are limited by your friend group size

Local Multiplayer Screen Types

- Split Screen
- Shared Screen
- Hotseat
  - Pass the controller to other players to take their turn
  - Ex: Civilization VI
- Asymmetric
  - Some players use different controllers or displays
  - Ex: Luigi's Ghost Mansion
- Voronoi Split Screen
  - Screen transitions between split and shared dynamically
  - Ex: Lego Star Wars
- Local Network (LAN)
  - Everyone has their own platform and their own screen

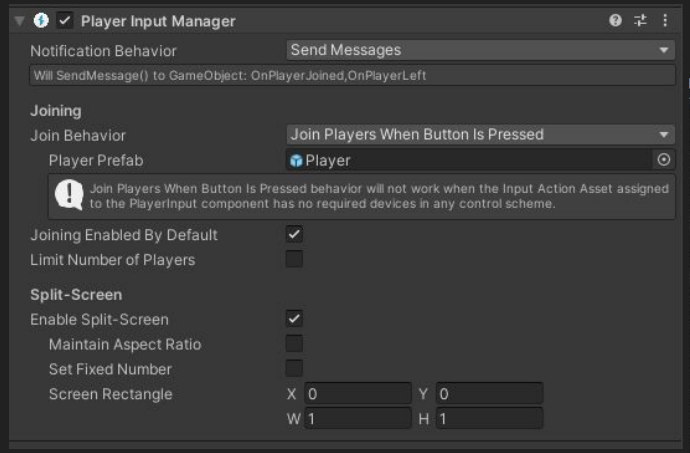
Local Multiplayer Screen Types (cont.)

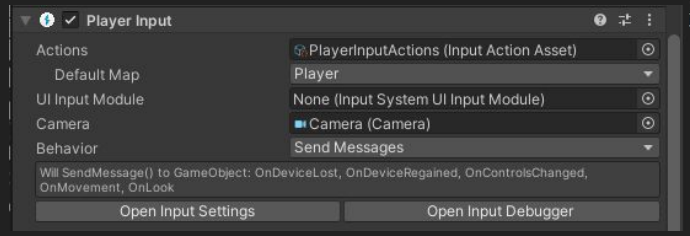
- Determining how to display the screen in a multiplayer game is not always simple
  - Depends on the type of game you are making
  - Ex: hotseat multiplayer might only be suitable for turn based games
  - Ex: split screen might not be suitable for competitive player versus player games
- Can be a challenging and interesting problem for game programmers
- Voronoi Split Screen:
  - https://youtu.be/tu-Qe66AvtY?si=pU9w8L0-GtVBOJ3w&t=1536

Local Multiplayer in Unity

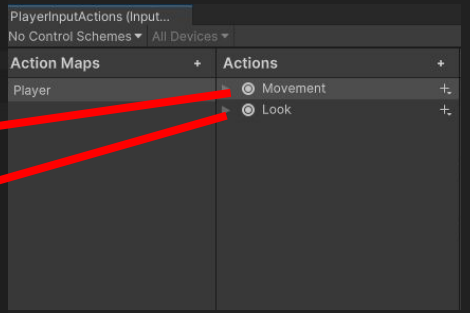
- Local multiplayer can be implemented in Unity using the new input system
- Can be implemented easily using:
  - PlayerInputManager (in Scene)
  - PlayerInput (on player prefab)
  - NOTE:
    - PlayerInput should have Behavior set to “Send Messages”
    - Custom script must have “OnX” public void functions with InputValue parameter
    - “X” is the name of the Action in the Action Map
- https://www.youtube.com/watch?v=g_s0y5yFxYg
- https://www.youtube.com/watch?v=lGxXQzE5Vu8

Local Multiplayer in Unity (cont.)






```
public class CubeController : MonoBehaviour
{
    public float velocity = 5f;
    public float lookSensitivity = 0.05f;
    private Vector2 movement;
    private Vector2 look;

    Unity Message | 0 references
    private void Start()...

    Unity Message | 0 references
    private void Update()...

    0 references
    public void OnMovement(InputValue value)
    {
        movement = value.Get<Vector2>();
    }

    0 references
    public void OnLook(InputValue value)
    {
        look = value.Get<Vector2>();
    }

    1 reference
    private void MoveAndRotate(Vector2 movementDelta, float rotationDelta)...
}
```



Local Multiplayer in Unity (cont.)

CubeController.cs

```
public class CubeController : MonoBehaviour
{
    public float velocity = 5f;
    public float lookSensitivity = 0.05f;
    private Vector2 movement;
    private Vector2 look;

    private void Start()
    {
        var playerInput = GetComponent<PlayerInput>();
        Debug.Log($"Player #{playerInput.playerIndex} has joined");
    }

    private void Update()
    {
        MoveAndRotate(movement, look.x);
    }

    public void OnMovement(InputValue value)
    {
        movement = value.Get<Vector2>();
    }

    public void OnLook(InputValue value)
    {
        look = value.Get<Vector2>();
    }

    private void MoveAndRotate(Vector2 movementDelta, float rotationDelta)
    {
        var position = transform.rotation * (velocity * new Vector3(movementDelta.x, 0f, movementDelta.y)) + transform.position;
        var initialY = transform.position.y;
        position.y = initialY;
        transform.position = Vector3.Lerp(transform.position, position, Time.deltaTime);

        var rotation = transform.rotation.eulerAngles;
        rotation.y += lookSensitivity * rotationDelta;
        transform.rotation = Quaternion.Euler(rotation);
    }
}
```

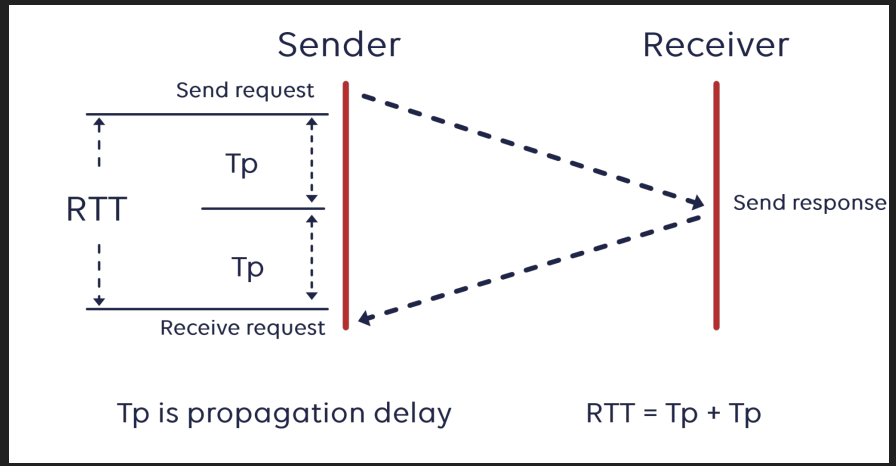
Online Multiplayer

- Online multiplayer lets you play games with people anywhere
- Requires a stable internet connection
- Allows you to play with many more people
  - How many? Technically, somewhere between 1 and 7,500+
  - https://en.wikipedia.org/wiki/Battle_of_B-R5RB

Ping and RTT

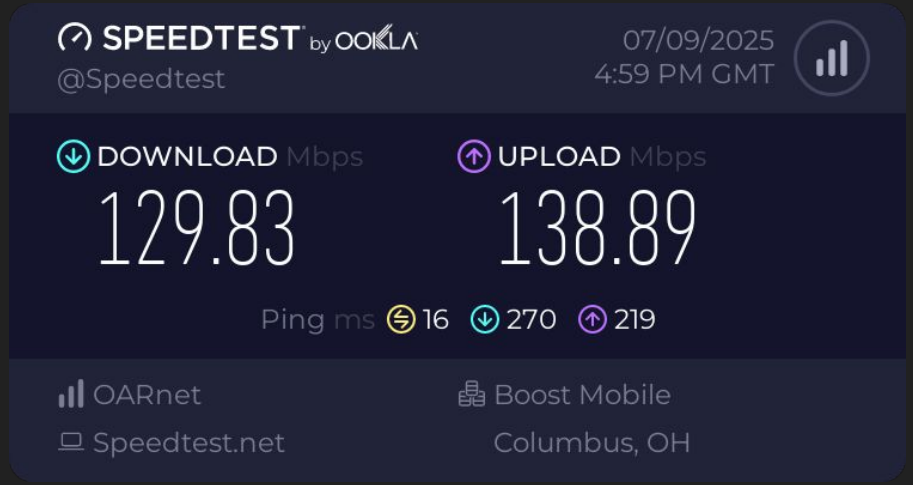
- Ping or Round Trip Time (RTT) is the amount of time it takes to send data and receive a response
- Generally, it takes half this time to send an update to a server
  - Less than 60 ms is best
  - Anything over 100 ms is noticeable

```
C:\Users\mgilson>ping ping-nae.ds.on.epicgames.com -n 100

Ping statistics for 44.192.143.240:
    Packets: Sent = 100, Received = 100, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 14ms, Maximum = 53ms, Average = 18ms
```



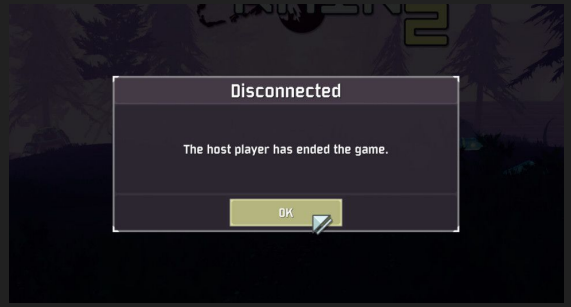
Bandwidth

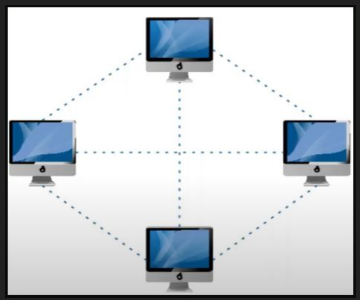
- Bandwidth is the amount of data a server can support
- The bandwidth required is different for every game
  - Ex: 2011 Minecraft has been tested around 100Mb/hr/user (0.027 Mbits/sec per user)
  - https://gaming.stackexchange.com/questions/22154/how-much-bandwidth-does-a-minecraft-client-use-in-smp
- Speed test: https://www.speedtest.net/
  - The example provided suggests I could host a Minecraft server with 4000+ players
  - Not true actually, my hardware, not bandwidth, is the limiting factor
  - The world record is 2622 players (?)



Online Multiplayer Architectures (P2P)

- Peer-to-peer (P2P)
  - Direct - players are directly connected
  - Player Hosted Client Server - one player acts as the host aka a listen server
- Relay Server - middleman between peers
  - DDOS Protection
  - Player’s IP addresses hidden
  - Steamworks
  - Epic Online Services



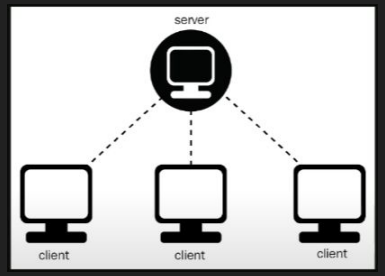


Online Multiplayer Architectures (P2P) (cont.)

- P2P benefits:
  - Cheap - you don’t have to pay for server hosting for your players
  - Simple - players use their game client to play the game and host the server (if NOT using direct P2P)
  - Robust - you don’t have to worry about the servers going down because there are no “servers” just players
- P2P downsides:
  - Trustworthiness - you can’t trust random players (unless a relay service is used)
  - Hacking - if players are the server, they can manipulate the game to do whatever they want (unsuitable for competitive or player versus player games)
  - Host advantage - the host has no latency and host has the power to end the game

Online Multiplayer Architectures (Dedicated Server)

- Dedicated Server
  - You run a server that is *dedicated* to keeping the players connected and simulating the game logic
  - This server is the authority of the game: what it says, goes
    - Prevents some player hacking/cheating

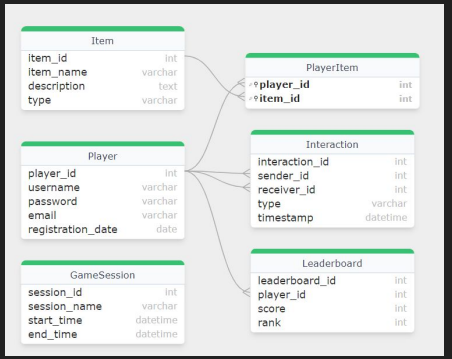


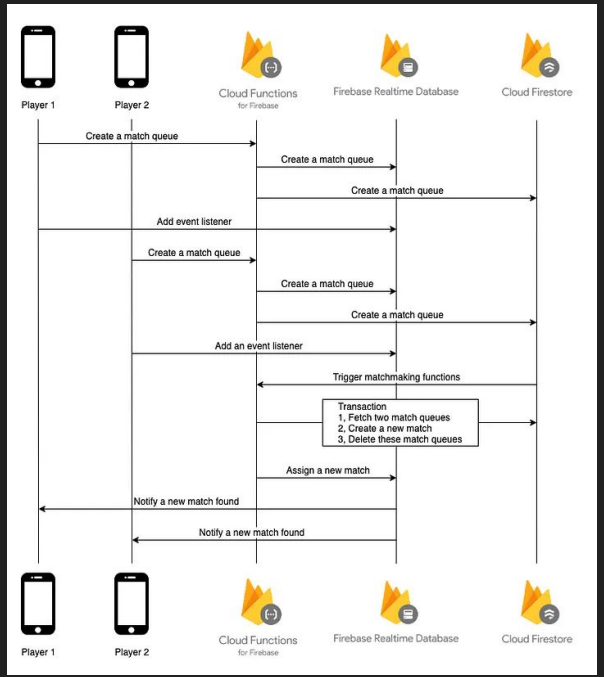
Online Multiplayer Architectures (Dedicated Server) (cont.)

- Dedicated Server benefits:
  - Performance - completely dedicated to keeping the connection/game running
  - Scale - if you have the hardware, you can run as many servers as you want
  - Security - certain hacking/cheating is no longer possible
- Dedicated Server downsides:
  - Cost - requires powerful expensive hardware to run the server
  - Complexity - requires a custom server program for keeping the game running and maintaining communication between all players
  - End of Life - when the servers are too expensive and are shut down, players lose the ability to play

Online Multiplayer Architectures (Database)

- Database model
  - Clients send and receive data from a database server
  - Server hosts the database, kind of like a “bulletin board”
  - Can be very cheap (free) if using something like Firebase
  - https://www.youtube.com/watch?v=6cX0AISOCI4



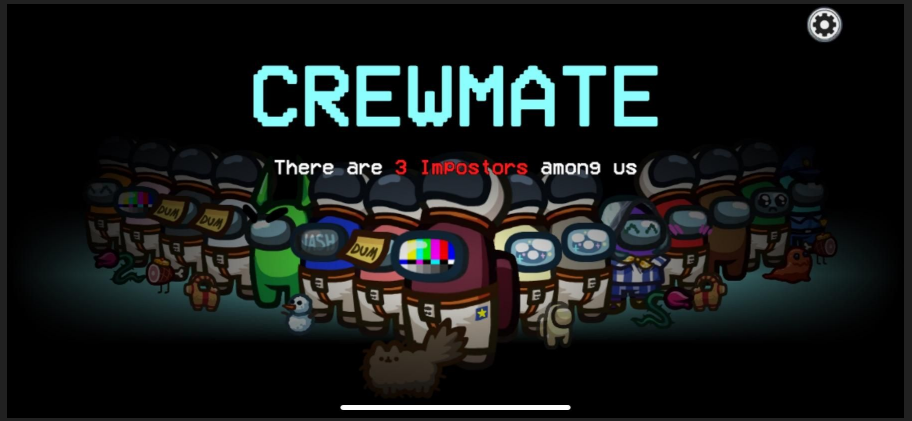


Online Multiplayer Architectures (Database) (cont.)

- Database model benefits:
  - Cheap - databases are very easy to run cheaply and requires minimal hardware
  - Scalable - databases can scale to 100,000’s of users
  - Simple - databases are just bulletin boards of information
  - Security - certain hacking/cheating is no longer possible
  - Asynchronous - players don’t have to stay connected all the time
- Database model downsides:
  - Debugging - unless you have a snapshot of the database when a user experienced a bug it could be hard to replicate the error
  - Simple - databases are not capable of doing advanced things like running a simulation of the game
  - End of Life - when the database is too expensive and are shut down, players lose the ability to play

Server Costs Estimate

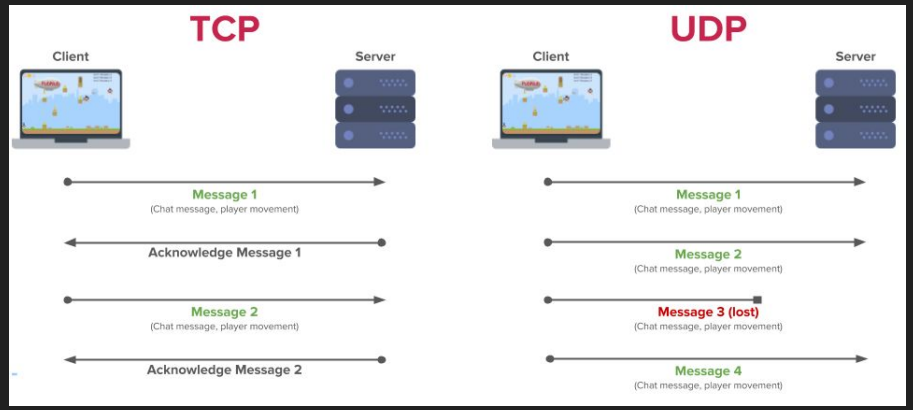
- Among Us:
  - Network Setup: Relay Server
  - Netcode: Hazel
  - Server: Linode
  - $5 per month supports 50 lobbies of 10 players, aka 500 players.
  - 3 matchmaking servers, 1 database server and 10-100 relay servers
  - This setup costs $160 a month to support 25k concurrent players.



Source: https://www.reddit.com/r/gamedev/comments/xwnyga/multiplayer_networking_solutions/

Communication Protocols

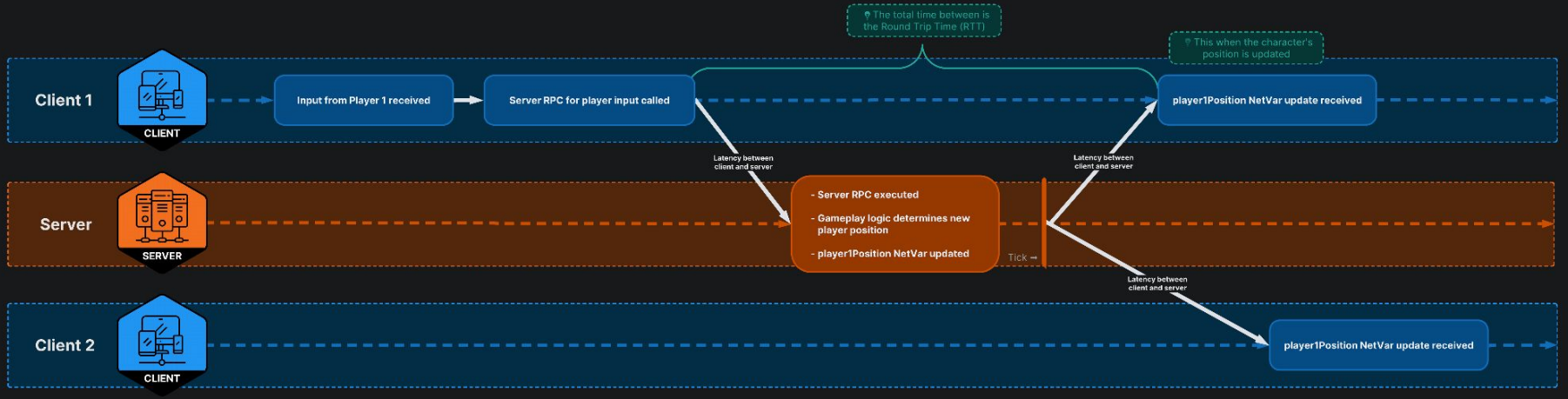
- TCP/UDP
  - IP address - apartment complex
  - Port - apartment number
  - TCP is guaranteed delivery with some overhead
  - UDP is not guaranteed delivery with more speed
- Remote Procedure Call (RPC)
  - The client asks the server to perform an action for them
  - Ex: client asks the server to move their character using RPC, server moves the character and returns the true position



Server Authoritative vs Client Authoritative

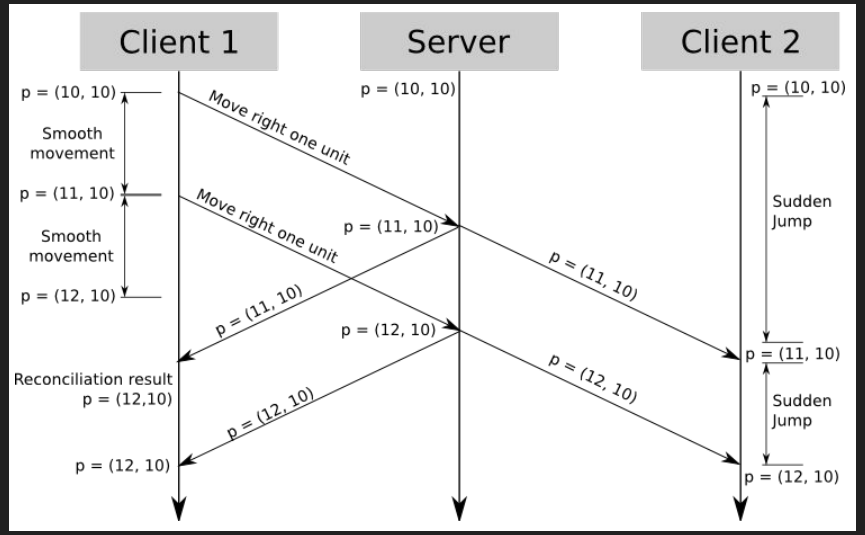
- Server authoritative
  - The dedicated server is the authority over the game logic
  - This prevents the players from hacking in certain ways
  - Players just send their controller inputs to the server and the server updates the game logic
- Client authoritative
  - The client is the authority over the game logic
  - This cannot prevent clients from doing whatever they want
    - When (not if) people hack your game
- Hybrid
  - Some things the client has authority over and some things the server has authority over

Server Authoritative



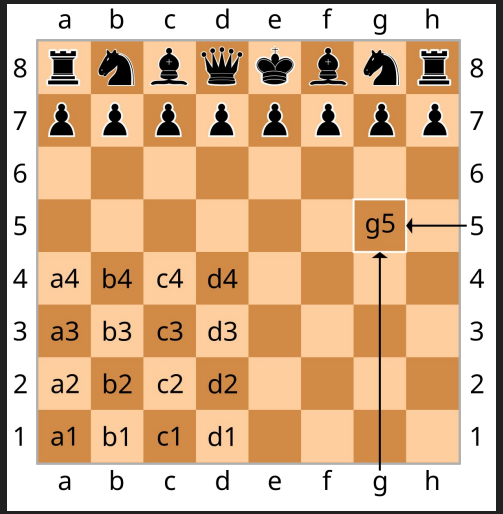
Client-Side Prediction

- If server communication is not fast enough or gets lost it will look jarring to the player
- If the client can predict what will happen next, gameplay will look much smoother
- Example: if you're playing a racing game, you know your current velocity, you can predict where you will be regardless of what the server says
    - Note: if your prediction was wrong, you will snap back to where the server says you are (rubberbanding)
    - https://www.youtube.com/watch?v=AQnF20Ez59M&ab_channel=3kliksphilip



Game State and Synchronization

- For chess, it's easy to create a game state:
  - The game's state is all of the pieces on the board and their positions
  - The game's state only changes when players move pieces
  - The only data that needs to be sent over the network is the state delta or change in state (not the whole state!)
  - (4 bits for piece + 6 bits for position) * 32 pieces = 320 bits = 40 bytes = 10 integers
- For other games, it may be much harder:
  - A game that uses physics is constantly updating even if the player is not doing anything
  - The transforms of every game object is probably too much data to send and receive 30+ ticks per second

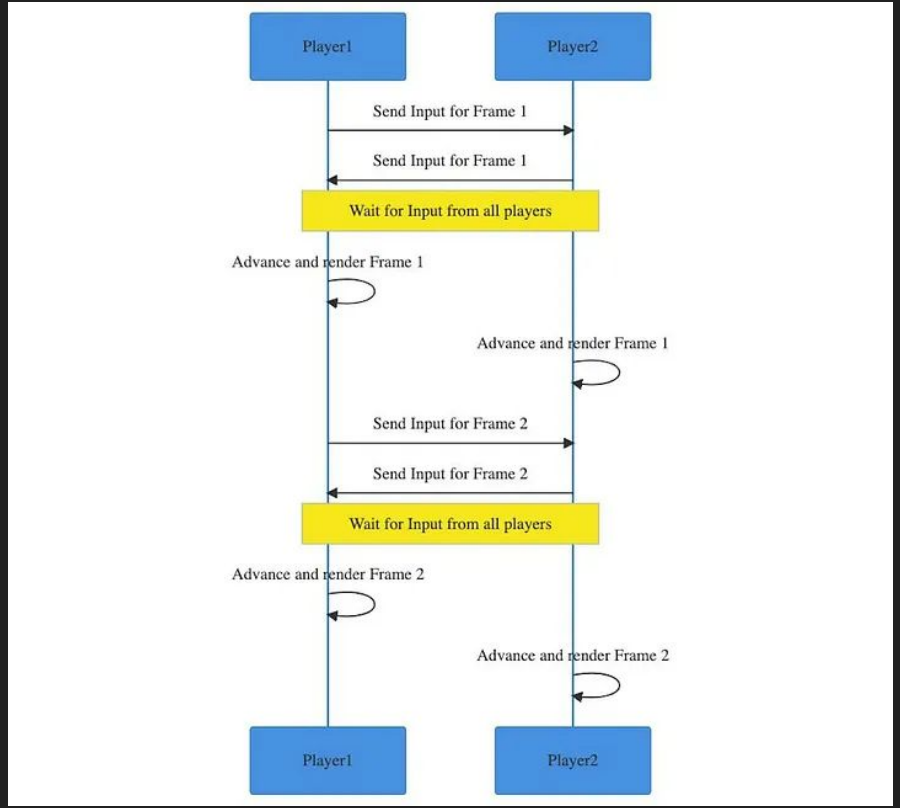


Game State and Synchronization (cont.)

- What happens if a player’s inputs are getting lost?
- What happens if the player has really high ping?
- What happens if we use UDP to get speed benefits but receive actions out of order?

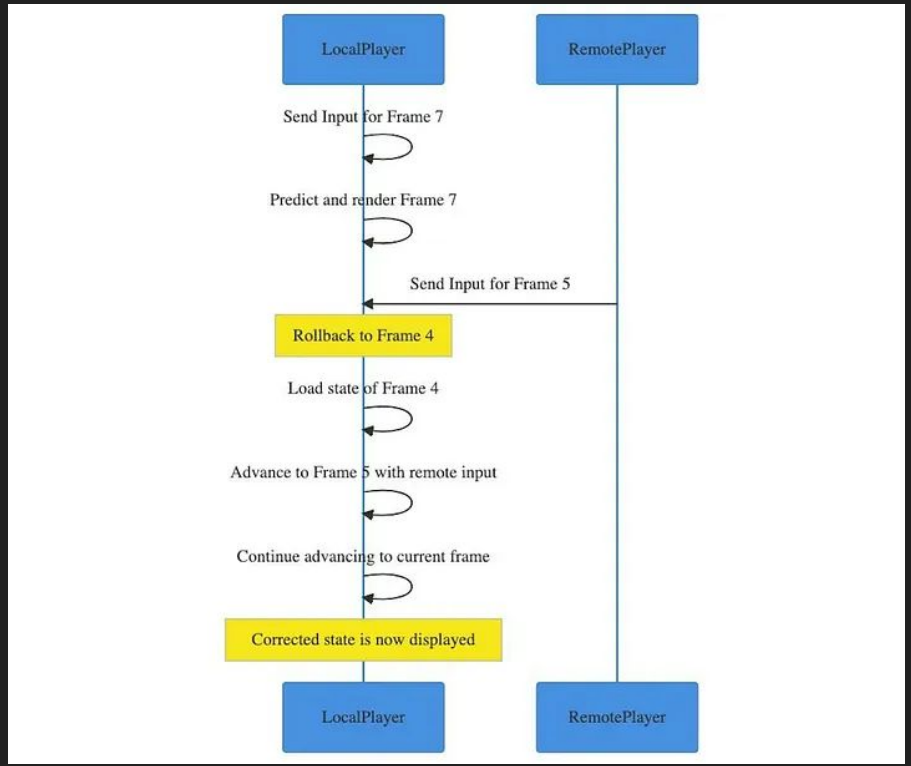
Lockstep Synchronization

- Lockstep synchronization forces the game to wait for input from all players before proceeding to the next tick
- The game engine must be deterministic across all platforms
- The game engine must also run at a fixed update rate



Rollback Synchronization

- Rollback synchronization is just like lockstep, except the game does not wait for the players
- Instead, if there is a delay, lag, or missed input, the other players “rollback” to a previous state



Snapshot Interpolation Synchronization

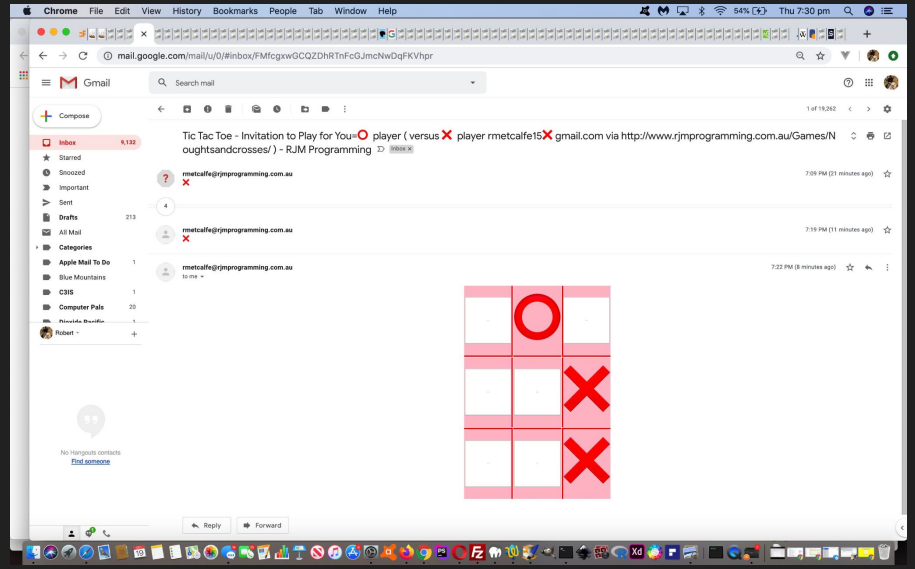
- Snapshot interpolation takes snapshots of objects in game and interpolates towards a predicted value
- If the predicted value and “true” value on the server disagree, value is teleported to “true” value
- Gameplay responds immediately to the player

Buffering

- Typically most games have some kind of buffer system
- This allows the game to continue running smoothly even when players have high ping
- The goal is to have this buffer be as small as possible and keep the player up to date as quickly as possible
- https://youtu.be/W3aieHjyNvw?si=AKTKgqTuF_GWQgkM&t=1478

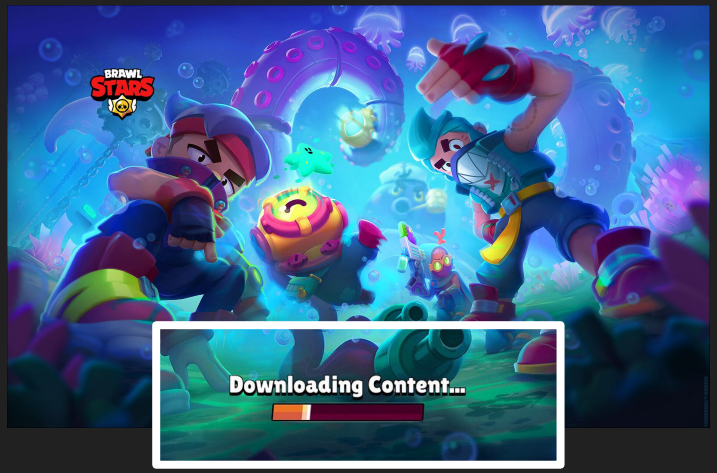
Asynchronous Online Multiplayer

- Asynchronous online multiplayer refers to games that don’t need heavy synchronization
  - Players don’t need to be constantly interacting with the game
- EX: Tic tac toe via email
- EX: Diplomacy played over weeks or months:
  - https://www.playdiplomacy.com/help.php



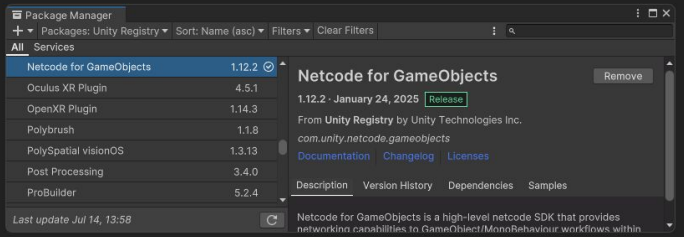
Live Service Games

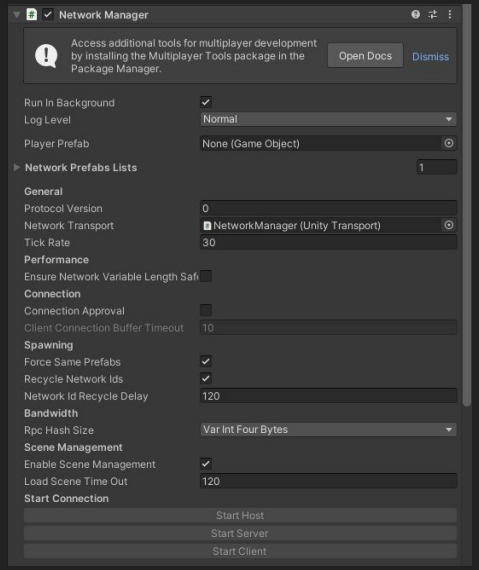
- Live service games are designed to be continually updated with new content after its initial release
- Goal is to keep players engaged over a long period through regular updates
- These updates are typically delivered with a patch and a content delivery network (CDN)
  - Can be self managed or use a service (like Steam or PSN)
  - Use AssetBundles in Unity: https://docs.unity3d.com/6000.1/Documentation/Manual/web-request-downloading-asset-bundle.html



Online Multiplayer in Unity (Netcode for GameObjects)

- Netcode for GameObjects is a Unity package that makes adding multiplayer functionality easier
  - Install using the package manager
- Add a NetworkManager component to Scene and select “Unity Transport”
- Add a NetworkStarter to the Network Manager to allow you to start a server or join as a client





Online Multiplayer in Unity (Netcode for GameObjects) (cont.)

- Use player input actions to allow you to select either hosting a game or to joining as a client

NetworkStarter.cs

```
using Unity.Netcode;
using UnityEngine;

[RequireComponent(typeof(NetworkManager))]
public class NetworkStarter : MonoBehaviour
{
    NetworkManager networkManager;
    private bool networkStarted = false;
    private PlayerInputActions playerInputActions;
    void Start()
    {
        networkManager = GetComponent<NetworkManager>();
        playerInputActions = new();
        playerInputActions.Enable();
    }

    // Update is called once per frame
    void Update()
    {
        if (playerInputActions.Player.Host.ReadValue<float>() != 0f && !networkStarted)
        {
            networkManager.StartHost();
            networkStarted = true;
        }

        if (playerInputActions.Player.Client.ReadValue<float>() != 0f && !networkStarted)
        {
            networkManager.StartClient();
            networkStarted = true;
        }
    }
}
```

Online Multiplayer in Unity (Netcode for GameObjects) (cont.)

- Create “ClientNetworkTransform”, inheriting from NetworkTransform and attach to player GameObjects
- Override “OnIsServerAuthorative” and return false

ClientNetworkTransform.cs

```
using Unity.Netcode.Components;

public class ClientNetworkTransform : NetworkTransform
{
    protected override bool OnIsServerAuthoritative()
    {
        return false;
    }
}
```

Online Multiplayer in Unity (Netcode for GameObjects) (cont.)

- In your script that controls the players:
  - Process controls if the GameObject is owned by the local player (not the remote player)
  - If not owned, disable the camera
  - Inherit from NetworkBehaviour

CubeController.cs

```
using Unity.Netcode;
using UnityEngine;
using UnityEngine.InputSystem;

public class CubeController : NetworkBehaviour
{
    public float velocity = 5f;
    public float lookSensitivity = 0.05f;
    private Vector2 movement;
    private Vector2 look;

    private void Start()
    {
        if (!IsOwner)
        {
            var otherCamera = GetComponentInChildren<Camera>();
            otherCamera.enabled = false;
            return;
        }
        var playerInput = GetComponent<PlayerInput>();
        Debug.Log($"Player #{playerInput.playerIndex} has joined");
    }

    private void Update()
    {
        if (!IsOwner) return;
        MoveAndRotate(movement, look.x);
    }

    public void OnMovement(InputValue value)
    {
        if (!IsOwner) return;
        movement = value.Get<Vector2>();
    }

    public void OnLook(InputValue value)
    {
        if (!IsOwner) return;
        look = value.Get<Vector2>();
    }

    private void MoveAndRotate(Vector2 movementDelta, float rotationDelta)
    {
        var position = transform.rotation * (velocity * new Vector3(movementDelta.x, 0f, movementDelta.y)) + transform.position;
        var initialY = transform.position.y;
        position.y = initialY;
        transform.position = Vector3.Lerp(transform.position, position, Time.deltaTime);

        var rotation = transform.rotation.eulerAngles;
        rotation.y += lookSensitivity * rotationDelta;
        transform.rotation = Quaternion.Euler(rotation);
    }
}
```

Online Multiplayer in Unity (Netcode for GameObjects) (cont.)

- ServerRPCs are executed on the server
- ClientRPCs are executed on clients
- NetworkVariable’s are used to synchronize variables between clients

NetworkMethods.cs

```
using Unity.Netcode;
using UnityEngine;

public class NetworkMethods : NetworkBehaviour
{
    private NetworkVariable<int> networkInteger = new();

    [ServerRpc]
    public void DoSomethingOnServer(int value)
    {
        networkInteger.Value = value;
    }

    [ClientRpc]
    public void DoSomethingOnClients(int value)
    {
        Debug.Log($"Value from server: {value}");
    }
}
```

Online Multiplayer in Unity (Netcode for GameObjects) (cont.)

- Assign the Player Prefab in the Network Manager
- Add a Network Object component to the Player Prefab