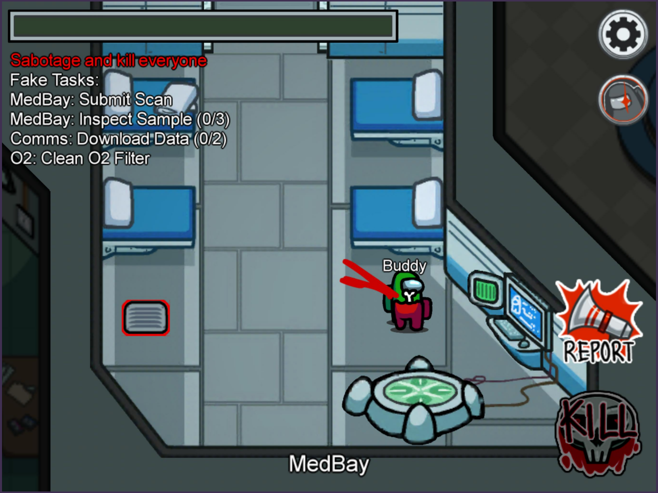
COMP280: Specialisms in Creative Computing

Networking in Unity

FALMOUTH UNIVERSITY

Multi-player Games


Sabotage and kill everyone
Fake Tasks:
MedBay: Submit Scan
MedBay: Inspect Sample (0/3)
Comms: Download Data (0/2)
O2: Clean O2 Filter

Multi-player Games

A word of caution

- I've chosen both my examples carefully
- Note the number of players
- (they're not MMOs or Battle Royale games!)
- Multi-player is hard, more players makes life harder!
- So stick to smaller games for now...

Basic Architecture

- We'll be considering a client-server model for this session
- True peer-to-peer for games are tricky
- Note: a 'server' could be one of the players
- It will be in our examples

State

- Before talk about networking, we should probably talk about the notion of *state*
- *State* is the information that makes up your game world
  - A door's state might be if it's position, and if it's open or closed
  - A player's state might be their position, how much health they have, what items they are carrying, etc...

The basic idea

- We want to make sure that the game's *state* is the same between users
- We can do this by ensuring that when something changes, we tell other players
- We usually do this by sending messages to an intermediary (a server) that then tells all the other players

Conflicts

- We're playing a game that features the ability to pick up objects
- Two players **both** try to pick up the same object at the same time
- Who has the object?

Basic Overview

A simplified view of networking for games:

1. Clients send request(s) / update(s) to the server
2. The server processes the request
3. The server lets the client(s) know the result
4. The clients update their state to match

nb. sometimes changes can happen on their own

Protocol

- The **Protocol** is the format of the messages between the clients and the server
- Can be / is usually game-specific
- Can be stateful (eg, move can only be sent after login)

Examples: Minecraft ′classic′ protocol, Designing Virtual Worlds, Overwatch Netcode Talk

By the end of today

- You should have a good understanding of the principles behind networking in Unity
- You should understand how to use Forge to build games
- You should have a simplistic multi player scene

Cubes!

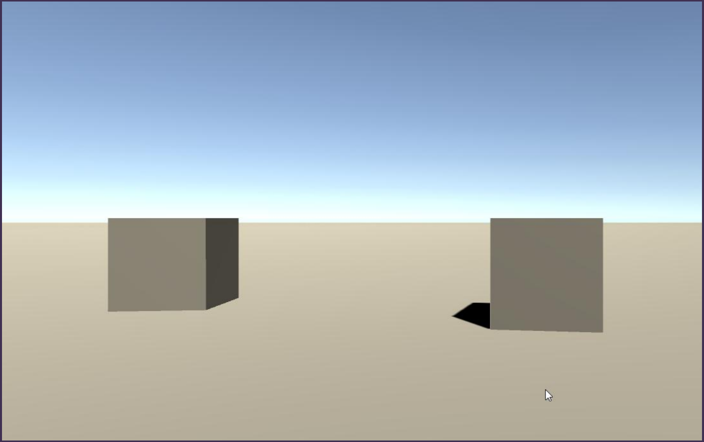


Figure: Basic Scene with Two Cubes

A simple networked application

- You can find the instructions for today on the learning space
- We'll be doing three (possibly 4 if you have time) activities to teach is the **Basics** of networking in Unity
- My example game is a little bare bones - that's so you can customise it

See Learning Space

Activity sheet on learning space

FALMOUTH UNIVERSITY

Activity 1 - Cubes

- Now you should have a working player controller with basic movement
- For the next part we'll make the clients have a cube to
- Questions?

Activity 2 - Two Cubes

- Every connected player should have a cube now
- They should be able to move independently
- **possible bug**: updating the transform directly can make physics unhappy

Activity 3 - RPC

- We've now seen how we can call methods on other peoples machines
- Very useful for dealing with anything that'd not happening every frame
- Careful about who can do what - possible security problems

Activity 4 - sync bugs

- Now we've seen what can happen when we don't sync stuff properly
- Question: would we ever want to do stuff on the clients? Why not do it all on the server?
- talking point: Rubber banding

Showcase?

Anyone want to share what they've made?

Networking in Unity

- Today we've built a simple networked application
- Syncing, creating objects and RPC are the cornerstones of our networking activities (at least for Unity...)
- Play with what you've learnt today and see what you can build!