Dedicated Server architecture

This book contains the basics for a Server-Client architecture where the Server is a dedicated server that runs all the game logic, and the clients mostly just send their actions, with the server deciding whether or not they're valid and replying with the updated game state.

Initial Setup

Basic initial setup for a godot 3d game in godot 4.3 is as follows.

Set up a basic 3D scene (no multiplayer yet)

Player:

We add a main scene in which everything will be happening.

Create a new scene "player", change root node to CharacterBody3D to start making a very basic player. add a script to it - the script **already contains movement controls**, but it uses the arrow keys or controllers rather than WASD.

add a collisionShape3D node to it, and to that add MeshInstance3D. select a CapsuleShape for the CollisionShape and a CapsuleMesh for the MeshInstance3D. This gives our player collision, and makes it visible in the game scene.

also add a Camera3D to the player scene, and move it away from the player a bit so you can see it moving around.

Game World:

in the main scene, we will add a node3d "objects" or similar, to fill it with some objects to move around on.

add a StaticBody3D, and to that node add a CollisionShape3D and a MeshInstance3D. select box shape for both.

Copy this. One of them will be the floor (rename the node) and the other will be a random box we put in the world. Move the floor, and change it's Transform -> Scale to be much bigger, so we can use it as floor. move the box to somewhere on the floor.

Add Multiplayer:

Exporting a Server

To make a dedicated server, we'll first need a way to tell the game that it's running as server. We can do this by going to project -> export -> add.... and adding an export preset. Here, we're using Linux. Call it Linux Server, and under the Resources tab, select Export as dedicated server. For now, leave everything else as is.

This does 2 things: first, it replaces textures and 3d models by placeholders. since the server doesn't need them, we save space and performance. Second, it'll automatically make the exported executable run with --headless, so we can easily run it on servers without any graphical interface, just using the command line. We can also use this fact in our scripts:

```
# Note: Feature tags are case-sensitive.
if OS.has_feature("dedicated_server"):
	# Run your server startup code here...
	pass
```

this info comes from the godot tutorials here:

https://docs.godotengine.org/en/stable/tutorials/export/exporting_for_dedicated_servers.html

For exporting on commandline, here's the full command:

godot --headless --export-debug "Linux Server" ./godot-server-client-arch.x86_64

note: this works if the current working directory is the project. otherwise, add the full path to the godot project file.

Connecting client to server

We add the following code to the script that runs at startup (here connected to the main node). This initializes the networking, the server exposes it's port and clients are able to connect. in this simple example, we use fixed ports and just connec to localhost.

```
# multiplayer server options
const PORT: int = 1117
const MAX_CLIENTS: int = 5

# multiplayer client options
const IP_ADDRESS = "127.0.0.1"
```

```
# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	if OS.has_feature("dedicated_server"):
		var peer = ENetMultiplayerPeer.new()
		peer.create_server(PORT, MAX_CLIENTS)
		multiplayer.multiplayer_peer = peer
	else:
		var peer = ENetMultiplayerPeer.new()
		peer.create_client(IP_ADDRESS, PORT)
		multiplayer.multiplayer_peer = peer
```

Now, when we build and start the dedicated server, it'll automatically listen to the set port on the local computer. (note: for connections to work across different networks, such as the internet, we'll need to add port forwarding and maybe firewall rules, otherwise they're blocked. this is out of scope for this tutorial.)

when we start the client, it automatically tries to connect to the local server. however, we don't really see anything happen, so let's add some debug messages via connecting various Signals (todo: link to a page explaining signals). Here's the finished code, built in a similar way as the official tutorial:

https://docs.godotengine.org/en/stable/tutorials/networking/high_level_multiplayer.html

```
extends Node3D

# multiplayer server options
const PORT: int = 1117
const MAX_CLIENTS: int = 5

# multiplayer client options
const IP_ADDRESS = "127.0.0.1"

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	if OS.has_feature("dedicated_server"):
		print("server starting... 2")
		var peer = ENetMultiplayerPeer.new()
		var error = peer.create_server(PORT, MAX_CLIENTS)
		if error:
			print("some error: " + error)
		multiplayer.multiplayer_peer = peer
```

```
	# signals?
	multiplayer.peer_connected.connect(_on_player_connected)
	multiplayer.peer_disconnected.connect(_on_player_disconnected)
	else:
	var peer = ENetMultiplayerPeer.new()
	var error = peer.create_client(IP_ADDRESS, PORT)
	if error:
		print("some error: " + error)
	multiplayer.multiplayer_peer = peer
	multiplayer.connected_to_server.connect(_on_connected_ok)
	multiplayer.connection_failed.connect(_on_connected_fail)
	multiplayer.server_disconnected.connect(_on_server_disconnected)
	


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta: float) -> void:
	pass



func _on_player_connected(id):
	print("player connected. id: " + str(id))
	
func _on_player_disconnected(id):
	print("player disconnected. id: " + str(id))
	
func _on_connected_ok():
	print("client connected")

func _on_connected_fail():
	print("client failed to connect")

func _on_server_disconnected():
	print("server disconnected.")
```

now we see what's happening, and get error messages if connections fail or drop.

We still can't do anything - so let's move on to the next step: spawning things. (todo: insert link)

Spawning things

Player

basics

in godot, we usually need to define what things are synchronized between server and client. for spawning, this is done with MultiplayerSpawner nodes.

In our main scene, add a node3d called "players". this is so we keep things organized - every player we spawn will be a child of that node. also add a MultiplayerSpawner (preferably not under players, as it is not a player). In the node properties of the MultiplayerSpawner, we need to set 2 things: where the spawned nodes shall appear -> Spawn Path, set this to the players node we just created, and what scenes it's allowed to spawn. this spawner is for players, so in it's Auto Spawn List, select the player scene we've created in initial setup. this tells the spawner exactly what scene to put where.

However, it doesn't actually contain any spawning logic - all it does is make sure that if the multiplayer authority (todo: link to explain authority) spawns the correct scene at the correct location, this same scene is replicated to any clients.

> NOTE: for the spawner to work, you need to use add_child(instance, true) to give the node a proper name, otherwise the spawner will refuse to work.

to see it working, add the following to the _on_player_connected(id):

```
func _on_player_connected(id):
	print("player connected. id: " + str(id))
	var player_scene = preload("res://player.tscn")
	var player = player_scene.instantiate()
	$players.add_child(player, true)
```

however, the players now spawn inside each other and physics probably freaks out. let's add:

Spawn Points

we can add a node "SpawnPoints" to our main scene, and inside it add points called 1, 2, 3 and so on. move them around a bit. then, we'll need to assign players a number - by default, their ID looks like this, a random number: player connected. id: 834055311
player connected. id: 271090083

for now, we just put a random selection in. (todo - fix spawn point and player ids)

however, even spawning in different places, physics freaks out - turns out, we did not spawn players correctly yet - we need to know authority, to know which client controls which player scene.

Authority