Tutorial implementasi komunikasi Godot & python

Tutorial ini hanya berisi code dan penjelasan dari code. Pada tutorial ini godot dan python tidak saling menunggu data yang dikirimkan melalui stream (TCP) tetapi dimulai (trigger) dari python. Jika pada REST API request dikirim oleh python lalu diberi response dari godot, namun berberda protokol dan data yang dikirimkan dimana REST API biasanya JSON sementara stream bisa apa saja selama mekanisme parsing data tepat.

A. Inisiasi TCP Godot
   a. Buat script untuk fungsional TCP (tidak perlu ada _**process** dan _**ready**

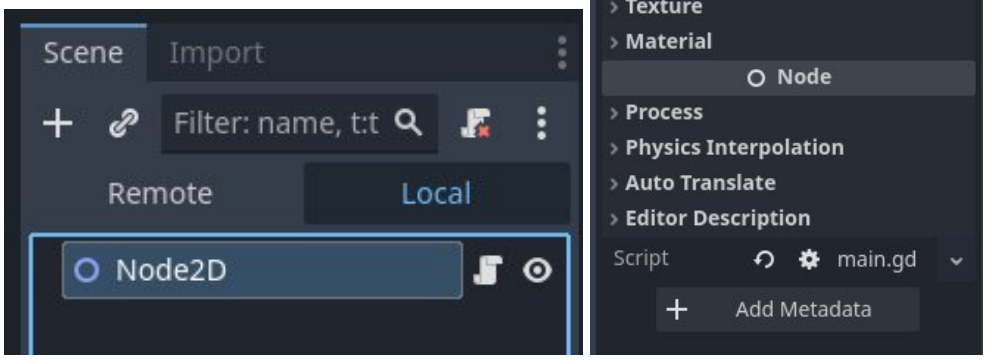
```
extends Node

class_name TcpChannel

var server := TCPServer.new()
var client
var buffer := ""

func start_server(port: int = 5555) -> bool:
    if server.listen(port) != OK:
        print("✖  Failed to start TCP server on port ", port)
        return false
    print("📡 Server listening on port ", port)
    return true
```

   b. Buat Scene berisi node dan terhubung ke sebuah script



   c. Inisiasi script TCP pada script scene

```
extends Node2D
var tcp_channel = TcpChannel.new()

var GODOT_RL_PORT = 7001
func init_channel():
    tcp_channel.start_server(GODOT_RL_PORT)
```

```
func _ready() -> void:
    init_channel()
    print("server ready")
```

B. Menambahkan handling menerima data dari stream (buffer) pada Godot

Gunakan **signal** pada godot untuk dapat **emit** atau mengirimkan trigger yang dapat memanggil fungsi. Untuk menerima data dapat menggunakan **thread** atau **poll** tetapi masing-masing memiliki cara implementasi yang berbeda. Pada tutorial ini digunakan **poll** yang perlu dipanggil terus menerus atau pada godot dipanggil pada _**process**.

Client digunakan untuk mengacu ke siapa yang sedang terhubung ke TCP (pada kasus ini python). Data yang digunakan JSON namun dikirim dalam bentuk **bytes**. Sehingga perlu dikonversi **byte -> string -> JSON**.

```gdscript
signal received_json(data: Dictionary)

func poll():
    if server.is_connection_available():
        client = server.take_connection()
        print("🔗 Client connected")

    if client != null:
        # Check if the client is still connected
        if client.get_status() == StreamPeerTCP.STATUS_NONE:
            print("⚠ Client disconnected")
            client.close()
            client = null
            buffer = ""
            return  # Exit early

        if client.get_available_bytes() > 0:
            buffer += client.get_utf8_string(client.get_available_bytes())
            var json = JSON.new()
            var error = json.parse(buffer)
            if error == OK:
                var data_received = json.data
                emit_signal("received_json", data_received)
                buffer = ""  # Clear buffer after successful parse
```

Untuk dapat menerima data dari poll gunakan **trigger signal** yang sudah dibuat pada scene script dengan **connect**. Signal “received_json” yang di emit saat poll menerima data akan secara otomatis memanggil **_on_received_json**.

```gdscript
func init_channel():
    tcp_channel.start_server(GODOT_RL_PORT)
    tcp_channel.connect("received_json",Callable(self,"_on_received_json"))

func _on_received_json(data: Dictionary):
    print(data)
    tcp_channel.send_json({"response" : "hello"})
```

C. Menambahkan fungsional mengirim data ke stream TCP pada Godot

Tergantung dari data yang mau dikirimkan baik hanya berupa angka, string atau seperti pada kasus ini JSON karena data yang dikirimkan terdiri dari beberapa variabel atau berbentuk *data structure*. Untuk mengirimkan data ke stream yang terus menerus mengirimkan data diperlukan **delimiter** untuk menandai pemisah data, dapat menggunan **“\n”** atau lainnya. Godot dapat mengirimkan string secara langsung namun dari percobaan yang dilakukan data yang dibaca oleh python memiliki tambahan berupa header sehingga perlu di cleansing. Sehingga disini digunakan **put_data** yang mengirimkan bytes.

```
func send_json(data: Dictionary):
    if client:
        var json = JSON.stringify(data)
        client.put_data(json.to_utf8_buffer() + "\n".to_utf8_buffer())
```

```
# Signature Help: gdscript
func StreamPeer.put_data(data: PackedByteArray) -> int
    func StreamPeer.put_data(data: PackedByteArray) -> int
```

D. Implementasi Python connect ke TCP Godot

Untuk dapat terhubung ke TCP yang sudah diinisiasi pada godot, python dapat menggunakan **socket**.

```
import socket
import json

class TCPClient:
    def __init__(self, host="127.0.0.1", port=5555):
        self.host = host
        self.port = port
        self.socket = self._connect()

    def _connect(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((self.host, self.port))
        return s
```

Lalu diinisasi pada script python lain atau main script.

```
from tcp_client import TCPClient

client = TCPClient(port=7001)
```

E. Implementasi Python menerima data melalui TCP dari Godot

Untuk dapat menerima data dari stream yang dikirimkan godot sama hal nya seperti poll yang diimplementasikan pada godot, dilakukan loop terus menerus dan berhenti berdasarkan kriteria yang dibuat (misal ketika ditemukan **delimiter**). Bergantung dari delimiter yang digunakan, loop yang digunakan mengambil data dari stream akan berbeda. Ditambahkan timeout untuk tidak menunggu terlalu lama, tetapi jika data yang dikirimkan cukup besar mungkin perlu mengubah cara menerima data.

```python
def receive(self, timeout=5):
    self.socket.settimeout(timeout)
    buffer = b""
    try:
        while b"\n" not in buffer:
            part = self.socket.recv(4096)
            if not part:
                raise ConnectionError("Socket connection closed by server")
            buffer += part
    except socket.timeout:
        return None
    finally:
        self.socket.settimeout(None)

    data_str = buffer.decode("utf-8").strip()
    if not data_str:
        return None

    try:
        return json.loads(data_str)
    except json.JSONDecodeError as e:
        print(f"(⚠ JSON decode error): {e} | Raw: {repr(data_str)}")
        return None
```

F. Implementasi Python mengirim data melalui TCP ke Godot

Dengan menggunakan **socket** pada python dapat mengirimkan pesan atau data ke data stream. Sama halnya dengan implementasi kirim data dari godot, pada python juga diperlukan **delimiter** untuk memisahkan antar data.

```python
    def send(self, data):
        message = json.dumps(data).encode("utf-8")
        self.socket.sendall(message + b"\n")  # newline delimiter
```

```python
from tcp_client import TCPClient

client = TCPClient(port=7001)

client.send({"hello": "world"})
response = client.receive()
print("Response:", response)
client.close()
```