gRPC communication for Unity

Author: Josip Josifovski josip.josifovsk@tum.de

The provided example includes the following:

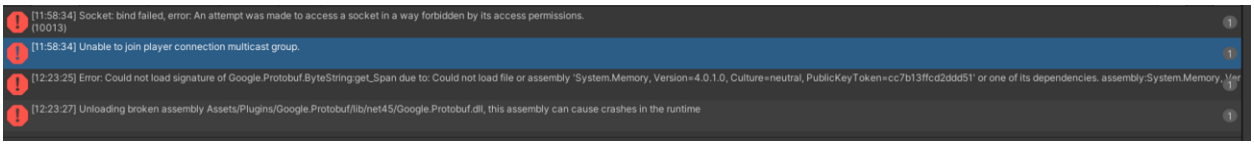
1) Definition of the gRPC service as it was specified in the RoboSkate API, the proto file is located in the **proto** folder
2) Extension of the simple ball game such that it implements a gRPC server from which the remote procedure can be called from external clients, the extended game is located in the **unity** folder
3) Python client that can be used for testing, once the game is running in Unity, the python client can call the remote procedures and control the game through the specified API. It is located in the **python** folder
4) grpc.tools.2.26.0 nuget package, that can be used to generate new c# scripts for the server in case the API needs to be modified. Located in **grpc.tools.2.26.0,** obtained externally as it was explained here

Unity version: **2019.4.8f1**

gRPC version: **grpc_unity_package.2.26.0-dev**

Operating system used for development and testing: **Windows 10**

Note: gRPC for Unity is in experimental phase, and the latest version (grpc_unity_package.2.35.0-dev202011181257.zip) was creating the following errors when it was included in the Plugins folder of the Unity project:



It seems there are several issues why the latest version doesn't work, the easiest workaround was to use an older gRPC version and that is why the used version for Unity is grpc_unity_package.2.26.0-dev. After using that version, the above errors related to gRPC were solved (the others are unrelated and unimportant). So if there is no special reason to go to the latest release, the 2.26 version of gRPC seems sufficient.

Using gRPC and the existing service in a new Unity project:

1) Extract the content of the Plugins folder in **grpc_unity_package.2.26.0-dev** to the Unity Plugins folder of your project. (Or simply copy the content of the NRPDummyBallGame in this case)
2) Copy the Service.cs and ServiceGrpc.cs scripts to your scripts folder (these scripts are auto-generated)

3) Copy the CommunicationServiceController.cs and modify it to serve your needs (this is the example class I implemented that starts the server on a specified port and waits for the remote calls, executes them and returns results from the game according to the RoboSkate API)
4) After all is prepared, you can run the game and it will start the server, now external clients can call the methods on the specified port and interact with the game
5) An example python client is provided (*python* folder) to use it for testing purposes (dependencies for python were easy to install as the python gRPC version is stable, I followed the official instructions)
6) After setting up the python side, when the unity game is running and the server works properly, once you run the python/client.py script, it should be able to connect to the game, initialize it, move the ball, get camera images etc.

Updating the gRPC service:
It is very probable that there will be changes in the API, because the methods might need modifications or new methods will be added. In that case, the following steps are needed:

1) Update the proto/service.proto file to include the modifications/new functionality
2) Unity side: Once the modifications are done, you need to create new Service.cs and ServiceGrpc.cs classes for your Unity project. You can do this by opening terminal in the *grpcPythonUnityCommunication* folder and executing the command below:

grpc.tools.2.26.0\tools\windows_x64\protoc.exe -I . --csharp_out=. --grpc_out=. --plugin=protoc-gen-grpc="Grpc.Tools.2.26.0\tools\windows_x64\grpc_csharp_plugin.exe" proto/service.proto

3) The above command will generate new Service.cs and ServiceGrpc.cs scripts, these need to be used to replace the old ones in the Unity project
4) Once this is done, modify the CommunicationServiceController.cs to include the functionality if needed
5) Python side: If you want to test the new functionality with the python client, you need to generate new classes for the python client also. This is explained in the official documentation. In my case, after activating the virtual environment with all dependencies in the terminal and navigating inside the proto folder I execute the command below:

python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. service.proto

6) This creates 2 files in the *proto* folder: *service_pb2.py* and *service_pb2_grpc.py.* These files should be used to replace the existing ones in the python/client folder.

Miscellaneous:

Some statistics about how fast was the communication on my machine in image transfer:

- Time it took for 1000 images: 147.2943754196167 (HD, 1920 x 1080) 6.8 FPS
- Time it took for 1000 images: 38.80050706863403 (500 x 500) 26.3 FPS (~8x smaller, ~4x speedup)
- Time it took for 1000 images: 24.21947169303894 (500 x 500) 41 FPS (just transfer, no rendering, no encoding to jpg and no copying to ByteString before transfer)