IOMapper: The Integration of Generalized Signal Control Mapping With the Godot Game Engine

Logan Murphy
Department of Computer Science
Dalhousie University
Halifax, Canada
lmurph@dal.ca

Joseph Malloch
Department of Computer Science
Dalhousie University
Halifax, Canada
jmalloch@dal.ca

***Abstract*—We present a novel software solution for dynamic control of properties in a simulated environment using custom signals and devices, enabling thorough creative control for contemporary media synthesis. The software tool is IOMapper, a plugin for the popular open-source Godot[1] game engine, serving as an implementation of the libmapper software library [1]. The portable nature of libmapper's network signals offers a basis upon which to easily integrate experimental input devices with Godot, creating a solid foundation for both formal study as well as the creation of art. IOMapper is a working tool, available publicly at https://github.com/lemurph/IOMapper.**

Keywords—art, experimental input devices, Godot game engine, media synthesis, software tool

I. Introduction

The need for a tool to ease the process of connecting novel human computer interaction (HCI) devices has been shown to be fulfilled by libmapper [1]. Among those who create experimental HCI devices are artists who are interested in a device specific to their creative needs. There is a vast community of live performers who incorporate visual elements in their musical and artistic performances. Many of these individuals cannot properly demonstrate their vision with a pre-existing controller and instead need a bespoke way to map signal data to the properties of the visuals. In this work we discuss the circumstances necessitating the implementation of libmapper into the Godot game engine, as well as the extensive portability that this implementation provides to those requiring a simple tool to quickly create an environment for testing experimental HCI devices.

II. Development Process

The decision to use the Godot game engine over another free engine such as Unity stems from two notable advantages that Godot has. The first advantage is the open-source nature of Godot. Since Godot is open source, this means that all of its features are free. Godot also encourages extension of the engine via C++, making the integration of libmapper uncomplicated. The second attraction of Godot comes from its ease of use. Godot lacks some features that Unity may have, but this means that Godot is far less complex. We expect a reasonable number of those using IOMapper to have limited experience with game engines, and Godot's uncomplicated design enables a great environment to quickly begin creating a project.

Beginning with no experience using a game engine, the development process began as an exploration into the capabilities of Godot. By experimenting and delving into the various example projects made available by users of Godot, an understanding of the user experience began to form. This understanding is fundamental in the development process, as the module would fail to be useful unless it matches the needs and expectations of Godot users. The official Godot documentation[2] was a valuable resource in learning how to use and connect the various systems available, as well as how a library such as libmapper may be bound to Godot's scripting language, GDScript.

We then encountered a slight impasse in trying to decide on an implementation of libmapper within Godot. When deciding between using a Godot extension called GDNative which would eliminate the need for recompiling Godot for every iteration of IOMapper, or alternatively developing a full module for the engine, we needed the expertise of Michal Seta. Michal had previous experience developing an Open Sound Control module for the Godot engine called gdosc[4], which was later ported to GDNative by a new maintainer. Meeting with Michal gave us the insight necessary to conclude that a full module would be almost indisputably superior to the GDNative approach based on its advantages with GDScript documentation integration and the advantageous community support that surrounded module development over the GDNative approach.

III. Structure of the Module

IOMapper follows a simple two-class structure as follows; the main IOMapper class which is instanced in GDScript as the libmapper device, and the signal class which can be instanced to store signals as GDScript variables and connected to other signals on the network. The signals can then be accessed for their data or used to transmit data to connected signals. There are a host of accessor and mutator methods available that can be called to alter or access the various properties or data of a signal. An example of a Godot script declaring a device with a signal can be seen in fig. 1.

Funding provided by the NSERC USRA

[1] https://godotengine.org/
[2] https://docs.godotengine.org/en/stable/index.html
[3] http://libmapper.github.io/
[4] https://github.com/djiamnot/gdosc

```
extends Node2D

# Create the device and store it in a variable
var device = IOMapper.new()
var test_signal
# Initialize device and signals
func _ready():
    device.init("test_device")
    test_signal = device.add_sig(IOMapper.INCOMING, "test_in", 1, IOMapper.FLOAT)

func _process(_delta):
    # The device must be polled in a process function ensure signals are up to date
    device.poll()

    var signal_data = test_signal.get_value_float()
    # test_signal can be connected to any outgoing signal on the local network and
    # signal_data can then be used for anything the user chooses
```

Fig. 1, A code snippet showcasing the creation of a device and adding a signal in Godot's GDScript using IOMapper.

The methods made available by IOMapper are more than sufficient to enable the use of Godot with any other libmapper-ready environment such as Max, Pure Data, SuperCollider, Processing and the Arduino or ESP32. Thanks to libmapper's language support, the ecosystem also enables users of C, C++, C#, Python and Java, as well as tools utilizing those languages (e.g Google's MediaPipe) a way to easily connect with Godot via IOMapper.

IV. Motivation and Usage

The *mapping* phase of interaction design for multimedia is often a complex task, only increasing in complexity with additional sensors [2]**.** The earliest works of HCI device design attempted to ease this complexity by focusing on equating the physical properties of different input devices to minimize the programming effort in substituting one device for another [3]. However, this approach becomes problematic for those interested in creating new HCI devices with idiosyncratic control methods. It is unreasonable to assume any person in need of a tool such as libmapper would be willing or able to learn how to use a new piece of software, then also implement that software into the tool they intend to use with the device. This hurdle necessitates the streamlined implementation of libmapper into various tools that creators are already familiar with, further expanding the libmapper ecosystem [1]. Since game engines are so versatile in use by nature, the implementation of libmapper into Godot provides multiple benefits for users of both tools.

First, as discussed in the introduction, artists who wish to incorporate an aspect of visual performance with their musical creations may wish to use a custom device to control the properties of the visuals and sound. This process would involve creating custom bindings from the tool to the Godot scripts, which is a tedious task that requires both time and sufficient programming knowledge. IOMapper eases this process of integrating with Godot by allowing the definition of custom signals that can transmit sensor values and the ability to assign a mapping expression to process the data, which incoming signals can then receive. The signal values that are received can be used to manipulate the properties of any Godot scene in any way the user chooses.

The second notable use case for IOMapper is as a research tool in the creation and visual testing of experimental HCI devices. The iterative nature of designing an interactive device may involve having to rewrite a large quantity of code for every iteration, possibly consuming valuable time. In this case, IOMapper uses libmapper to provide a way for researchers and HCI device creators to quickly define and map sensor values wirelessly over the local network, eliminating the need for a wired connection. The incorporation of this portability into a game engine provides a fully customizable environment which can be used to experiment with the control of a custom HCI device. IOMapper can serve as a valuable tool to a researcher by creating an environment for formal user studies as well as exercise the creative applications of a custom device.

V. Future Plans

In its current state, IOMapper has had limited feedback from users, especially those who are familiar with Godot. The next steps for IOMapper include gathering feedback from users of the module on features that may need to be implemented or tweaked. We are looking to gather feedback from the users on Godot's various communities such as their forums, sub-reddit and Discord group, all of which are active and supportive. As the module is focused on being a user-friendly tool, there is documentation available in the repository, as well as a wealth of documentation available for libmapper on its website and in its repository[3]. While IOMapper has thorough documentation including instructions and examples, this is also an area that is lacking in user feedback and refinement.

Additionally, there is a visual scripting language available for Godot called VisualScipt and it may be valuable to add IOMapper integration for that as well.

The next priority for IOMapper is porting it to other engines, starting with Unity. We are aware that those who are using game engines in their work are likely using Unity, and we would like to provide the flexibility of using IOMapper with whichever engine a user sees fit.

Acknowledgment

I thank Joseph Malloch for the supervision of the IOMapper project, as well as those responsible for creating and maintaining libmapper. I thank Matt Peachey for numerous helpful discussions, feedback, and contributions to IOMapper. I thank Michal Seta for an informed basis upon which to begin the development of IOMapper.

References

[1] J. Malloch, S. Sinclair, and M. M. Wanderley,"libmapper: (A Library for Connecting Things)," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, p.3087–3090, ACM, New York, NY, USA, 2013, ISBN9781450319522, doi:10.1145/2468356.2479617.

[2] J. Malloch, S. Sinclair, and M. M. Wanderley, "Generalized multi-instance control mapping for interactive media systems", IEEE Multimedia, p.39-50, January 2018, doi: 10.1109/MMUL.2018.112140028

[3] R. J. K. Jacob, L. E. Sibert, D. C. McFarlane, and M. P. Mullen "Integrality and separability of input devices," in *ACM Transitions on Computer-Human Interaction*, Volume 1, Issue 1, p.3-26, March 1994, doi:10.1145/174630.174631