OpEye^2

Technical documentation

Co-funded by the
Erasmus+ Programme
of the European Union

Collaborating partners

Aspaym Castilla y Léon

C/ Treviño 74
47008 – Valladolid
Spain
www.aspaymcyl.org
info@aspaymcyl.org

Center IRIS Ljubljana

Langusova ulica 8
1000 Ljubljana
Slovenia
www.center-iris.si
info@center-iris.si

Centre pour le développement des compétences relatives à la vue

17a, route de Longwy
L-8080 Bertrange
www.cc-cdv.lu
info@cc-cdv.lu

Lega del filo d'oro

Sede centrale di Osimo Via Montecerno, 1
60027 Osimo (AN)
Italy
www.legadelfilodoro.it
info@legadelfilodoro.it

Introduction

In this document, we explain how to set up the OPEYE 2 development environment and detail the structure of the project, providing insight into the methods and technologies utilized throughout the development process.

Resources used

This project made use of some pre-existing software. These key components include Unreal Engine 4.27.2 as the primary development platform, the *TobiiEyeTracker SDK* for eye-tracking capabilities, and the combination of *TCP Socket Plugin* and *EasyXML Parser* for seamless communication with the OpenGaze API. These resources have been instrumental in crafting the OPEYE 2 application, enabling its versatility and adaptability to various eye-tracking solutions.

Co-funded by the
Erasmus+ Programme
of the European Union

Setup

This guide assumes that you have already set up Tobii Experience and/or GazePoint and performed the necessary calibration process. In case these steps have not been performed yet, you can read up on them in our User Documentation.

To get started with the project clone the project from the github repository using the following command:

```
git clone https://github.com/Opeye-erasmus/OPEYE2
```

The basic setup requires the installation of Unreal Engine 4. To do this, download the Epic Launcher from the following link: https://www.unrealengine.com/en-US/download

Install and launch the Epic Launcher. Inside Epic Launcher, select “Unreal Engine” on the left side then click “Library” at the top and finally click on the “+” button next to ENGINE VERSION. Select version 4.27.2 and install it.

If you want to use the TobiiExperience plugin you need to apply to their SDK via their download form using this link: https://developer.tobii.com/software-downloads/. Once you have access to the plugin, download it and place it inside the plugin folder:

```
o3-communication-application\Opeye2\Plugins\TobiiEyetracking
```

Two more free plugins are needed for the setup to be completed. The two plugins are “TCP Socket Plugin” (https://www.unrealengine.com/marketplace/en-US/product/tcp-socket-plugin) and “EasyXML Parser” (https://unrealengine.com/marketplace/en-US/product/easyxmlparser). Since they are hosted on the UE4 marketplace, they should be downloaded automatically when you open the project.

Finally, after installing all the required elements, you can launch the project using one of two methods. Either navigate to the path where you used `git clone` and open the file `o3-communication-application\Opeye2\Opeye2.uproject` using Unreal Engine. Alternatively, you can launch Unreal Engine 4.27.2 from inside the Epic Launcher and browse to the location of the project to open it.

Co-funded by the Erasmus+ Programme of the European Union

Software architecture

OPEYE 2 was developed in Unreal Engine 4. The whole code architecture is based on Unreal Engine 4’s Blueprint system. Unreal Engine 4 Blueprints is a powerful visual scripting system that streamlines the development process by enabling rapid prototyping and iteration of game mechanics. Blueprints facilitate collaboration between designers and programmers, as it provides a clear, visual representation of game logic that is easy to understand even for those who were not initially part of the project. With well-commented Blueprints, the finished code becomes more maintainable and accessible, fostering better communication within the development team.

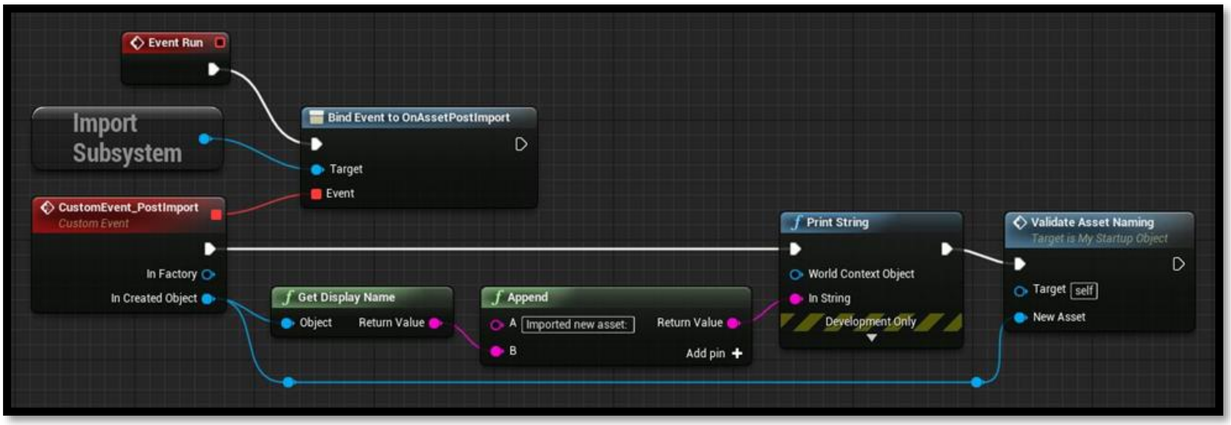


Figure 1 Blueprints Example from UE4 docs

By using Blueprints, developers can leverage the advantages of a visual scripting system that accelerates the development process while maintaining a high degree of flexibility. This system allows for easy modification or swapping out of certain parts of the code, ensuring a more adaptable and efficient development workflow. The intuitive nature of Blueprints not only enhances the overall development experience but also empowers developers to create a robust and polished end product.

To use Blueprints, open the Unreal Engine 4 editor and create a new Blueprint Class, selecting the appropriate parent class for your desired functionality. Once created, double-click the Blueprint to open the Blueprint Editor. Inside the editor, you can add nodes by right-clicking on the graph, searching for the desired function or variable, and selecting it from the list. Connect nodes by dragging wires from output pins to input pins, creating a flow of data and execution. Compile and save the Blueprint to apply the changes and see them reflected in your project.

The structure of the project is divided in folders as follows:

- `Art/`
- `Core/`
    - `Data/`
    - `Gameplay/`
    - `Interface/`
    - `MainMenu/`
    - `Misc/`
- `Levels/`

The project structure is organized into folders for easy navigation and better collaboration. You'll find materials, 3D models, and related assets in the **`Art/`** folder. Save-related logic is located in **`Core/Data/`**, while the main building blocks of the project, like Actors, Components, Pawns, and Spawners, are located in **`Core/Gameplay/`**. User interface elements displayed during application use can be found in **`Core/Interface/`**, and main menu logic is managed within **`Core/Mainmenu/`**. **`Core/Misc/`** holds a variety of elements, including buttons and miscellaneous text elements. Lastly, the **`Levels/`** folder contains the actual levels you'll experience in the application.

Apart from the main **`OpeyeIIContent/`** folder we have added **`iExpressIIContent/`**, **`StarterContent/`** and **`TS_PostProcess/`** folders to prevent having to refactor the code in case any plugin used in the projects changes their code structure.

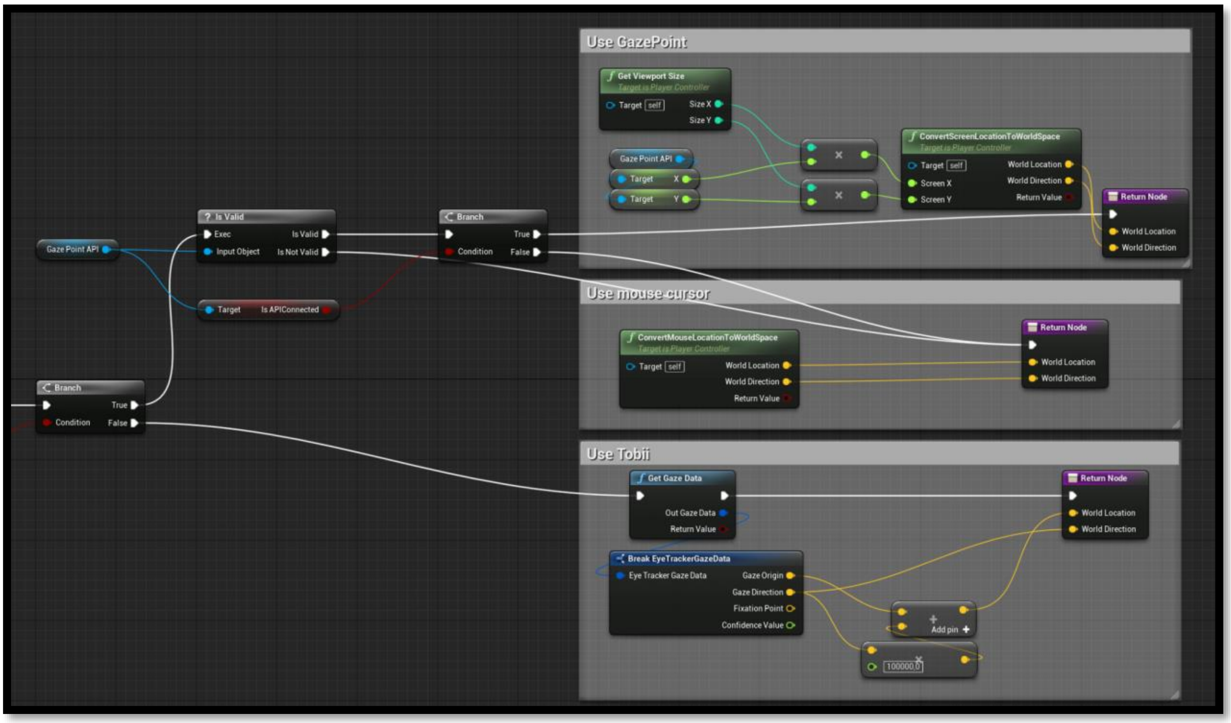
Eye Tracking

Eye tracking is primarily handled inside **`Core/Gameplay/Pawn/EyeTracker_Ctr`**. After some variables have been set when starting the application - the "Event Tick" takes over. To prevent overly stressing the host, an evaluation frequency can be adjusted so that the gaze estimation is only executed every x millisecond.

The function *Get Gaze Origin and Direction* gets used to trace a line between the user and the location where they are looking at in word coordinates. If this line hits an object, it gets communicated to the *3DTraveler Actor* to evaluate the next action.

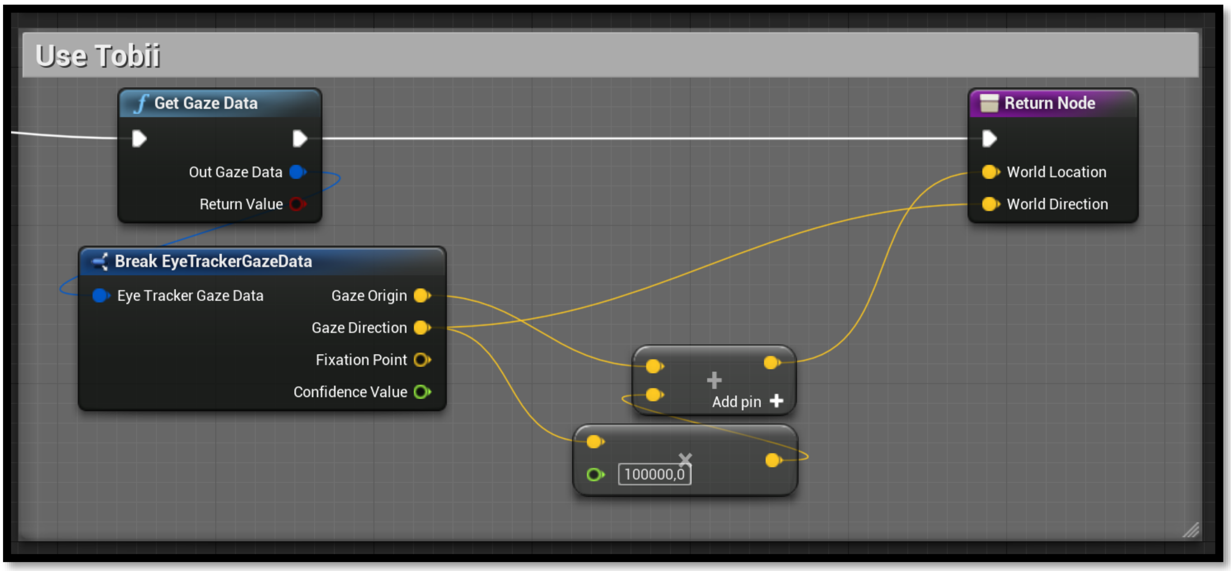
In a first step *Get Gaze Origin and Direction* detects which eye tracker should be used. If Tobii is not available, it will alternatively try to find a running OpenGaze API server. If no devices are detected at all, the function will use the mouse coordinates as a fallback option. This is especially helpful when debugging or simply when trying to demo the capabilities of the application without having to set up the full testing environment.

Co-funded by the
Erasmus+ Programme
of the European Union



Tobii Integration

When using the *Tobii Eyetracking* plugin with Unreal Engine 4, we can seamlessly integrate eye-tracking data into our project via Blueprints. We can access Tobii's eye-tracking functionality directly within UE4's Blueprint system. To begin, we create a new Blueprint or open an existing one. Inside the Blueprint Editor, we can access various Tobii-specific Blueprint nodes, such as "*Get Gaze Data*" or "*Get Combined Gaze Data*". These nodes allow us to retrieve eye gaze information, such as gaze direction, gaze origin, or focus object. If Tobii is detected these are used in the **`Eyetracker_Ctr`** and exported using the return node.



OpenGaze API

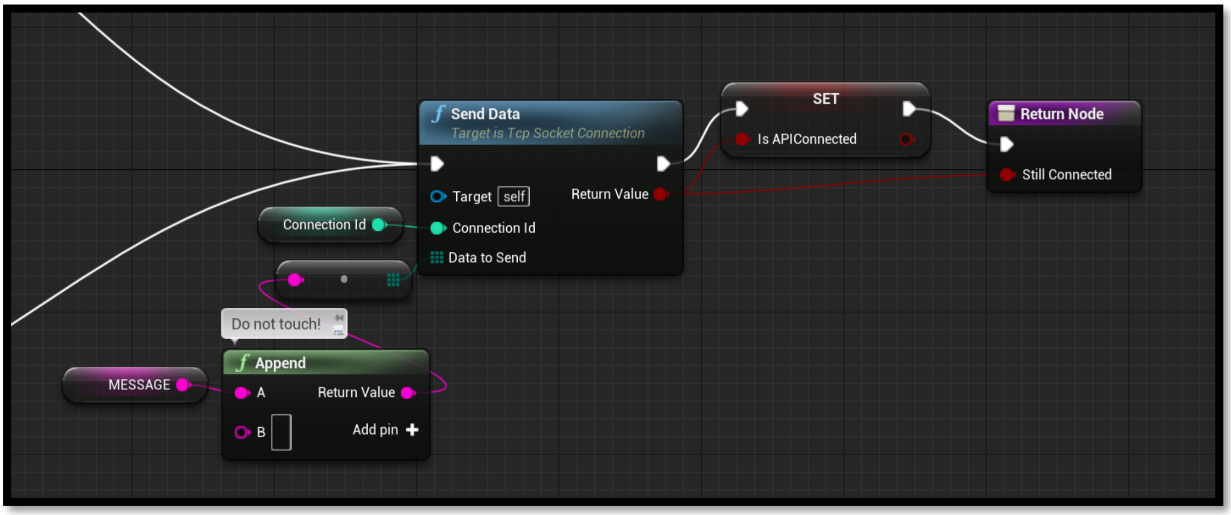
Compared to the finalized Tobii integration, the interaction with the OpenGaze API is more low-level. When working with the OpenGaze API to collect eye gaze data, we start by establishing a connection with the eye-tracking server. To do this, we create a TCP socket and connect it to the server's IP address, which is usually `127.0.0.1`, and the port number, which is typically `4242`.

Once connected to the server, we need to send specific XML commands to request eye gaze data. For example, we can send the command `<SET ID="ENABLE_SEND_DATA" STATE="1" />` to start receiving gaze data. If we want to receive data at a specific frequency, we can use the command `<SET ID="SAMPLE_RATE" VALUE="120" />`, which sets the sample rate to 120Hz. To define the data fields we wish to receive, we can use the command `<SET ID="ENABLE_SEND_PUPILSIZE" STATE="1" />`, which includes pupil size data in the stream. The full list of supported commands can be viewed here: https://www.gazept.com/dl/Gazepoint_API_v2.0.pdf

As the server starts sending gaze data, our application should listen for incoming XML strings on the TCP socket. Each string contains a single gaze data sample with the requested data fields. We can then parse these strings to extract relevant information such as gaze coordinates, pupil diameter, or eye movement events.

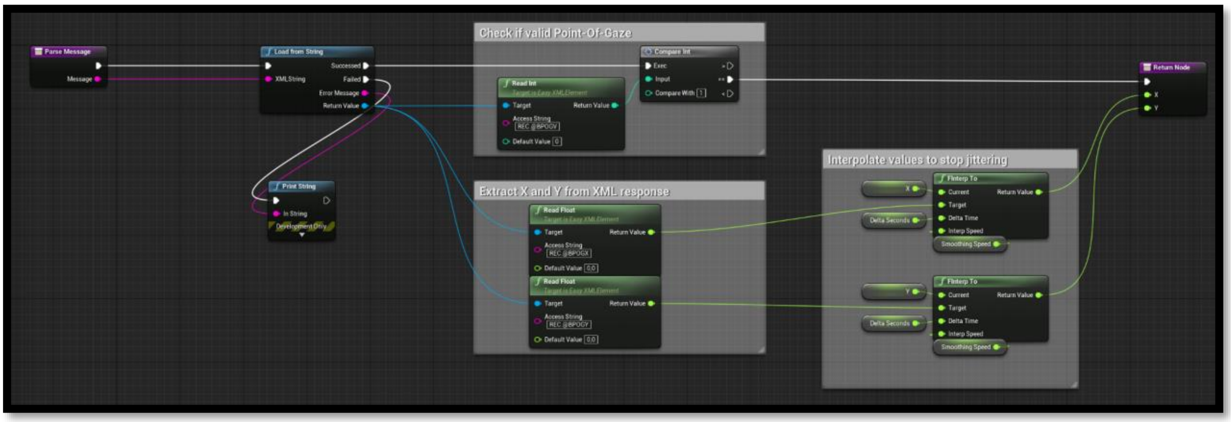
In UE4 we defined the *Send Message* function to send commands to the server. The messages need to be carefully crafted since the last two characters have to be “`\r\n`” namely carriage return (`CR`) and line-feed (`LF`) otherwise the OpenGazeAPI will ignore this message. It is important that these characters are not escaped before being sent to the server. The main issue with this was that most TCP socket plugins from the UE Marketplace were not able to craft such messages. They either added extraneous characters and/or escaped or trimmed the white-space characters that needed to be sent.

TCP Socket was the only free plugin with which we were able to communicate with the server. In the blueprints we added the white-space characters using the append node in which `A` contained the XML-command and the command delimiter (`\r\n`) in `B` was pasted from a text editor.



With the proper message composed and sent to the server, the server will straight away answer with the gaze data. This data gets parsed using the *Parse Message* function. Since the responses are also written in XML we used the *EasyXML Parser* plugin to extract the relevant variables. The typical reply from the server is composed as follows (depending on the initial commands send to the server):

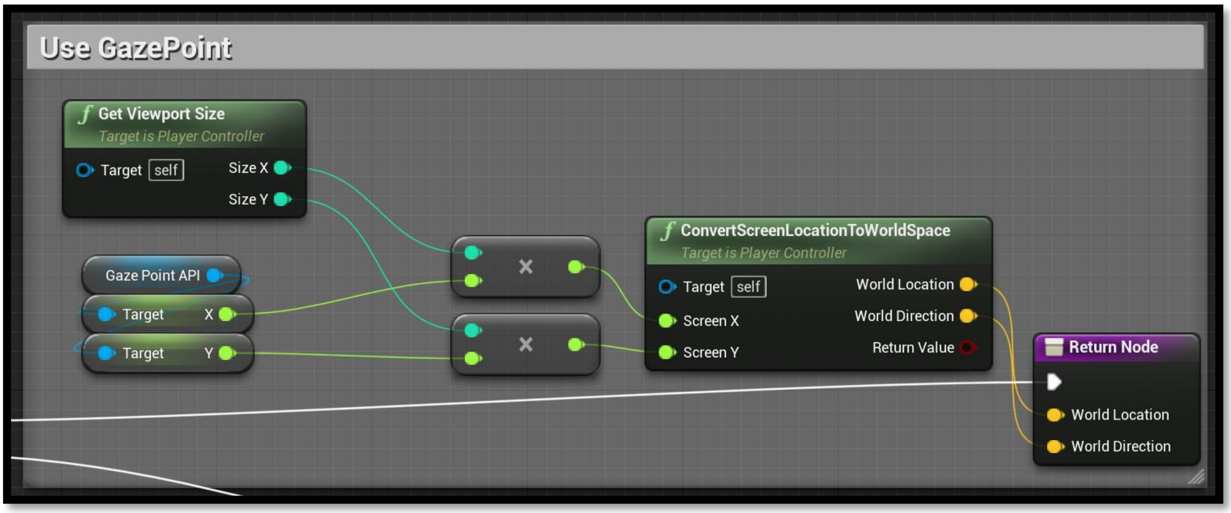
```
<REC BPOGX="0.38244" BPOGY="1.38776" BPOGV="0" />
```



Depending on the initial commands sent to the server the XML-attributes may change. In this case `BPOGX` and `BPOGY` tell us the best point of gaze data - which is the average of the left eye and right eye. If not it will send the best of either the left or right eye, depending on which one is valid. `BPOGV` tells us whether the values are valid or whether they should be ignored.

Finally, we interpolate X and Y to prevent major jittering and save and expose the values that we received. Back inside the **`EyeTracker_Ctr`** we multiply the values with the viewport size and convert the screen-location to world space.

Co-funded by the Erasmus+ Programme of the European Union

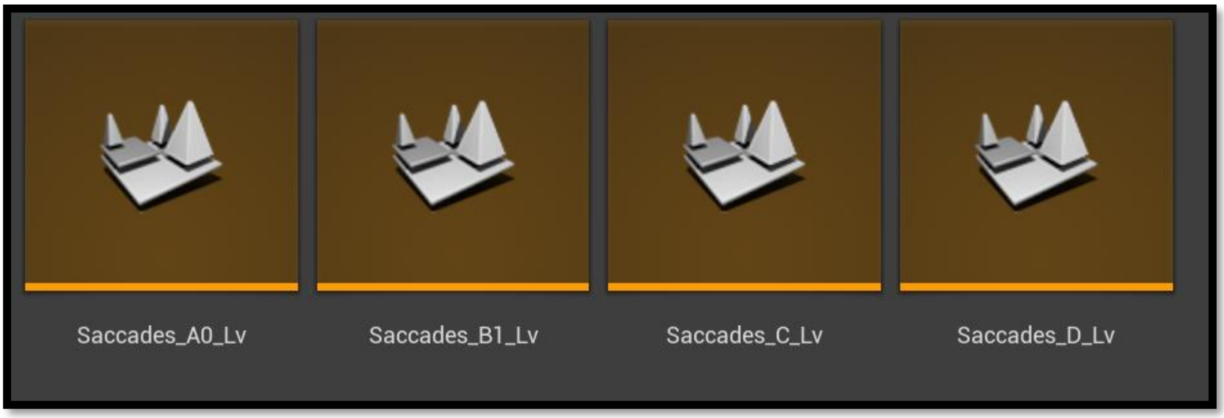


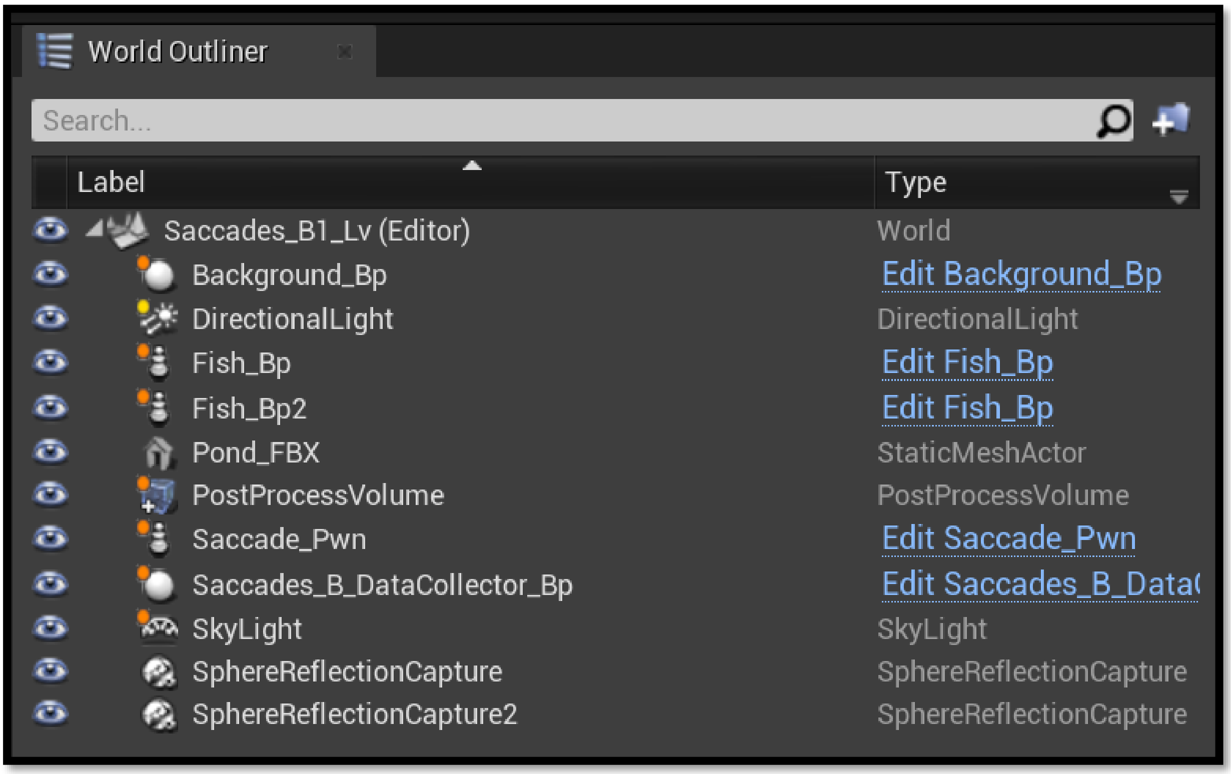
Levels

The various levels have very level dependent logics. Those are easier explored by downloading the project and investigating the associated blueprint files. To get a general overview of all the levels implemented in the application, check the user documentation.

Generally, it can be stated that all `_Lv` level files are located in **`Levels/`**. The various blueprints, pawns and actors are placed inside the level and will execute the actions defined in their blueprints when the level is started. By default, the blueprints that are used in the level can be found in the top right corner of the Editor window. The blue "Edit …" link can be used to jump straight to the blueprint. Alternatively, they can be found in **`Core/Gameplay/`**, inside their respective folders. Most of the level specific logic can be explored in the **`Core/Gameplay/Spawners/`** subfolder.

Co-funded by the Erasmus+ Programme of the European Union

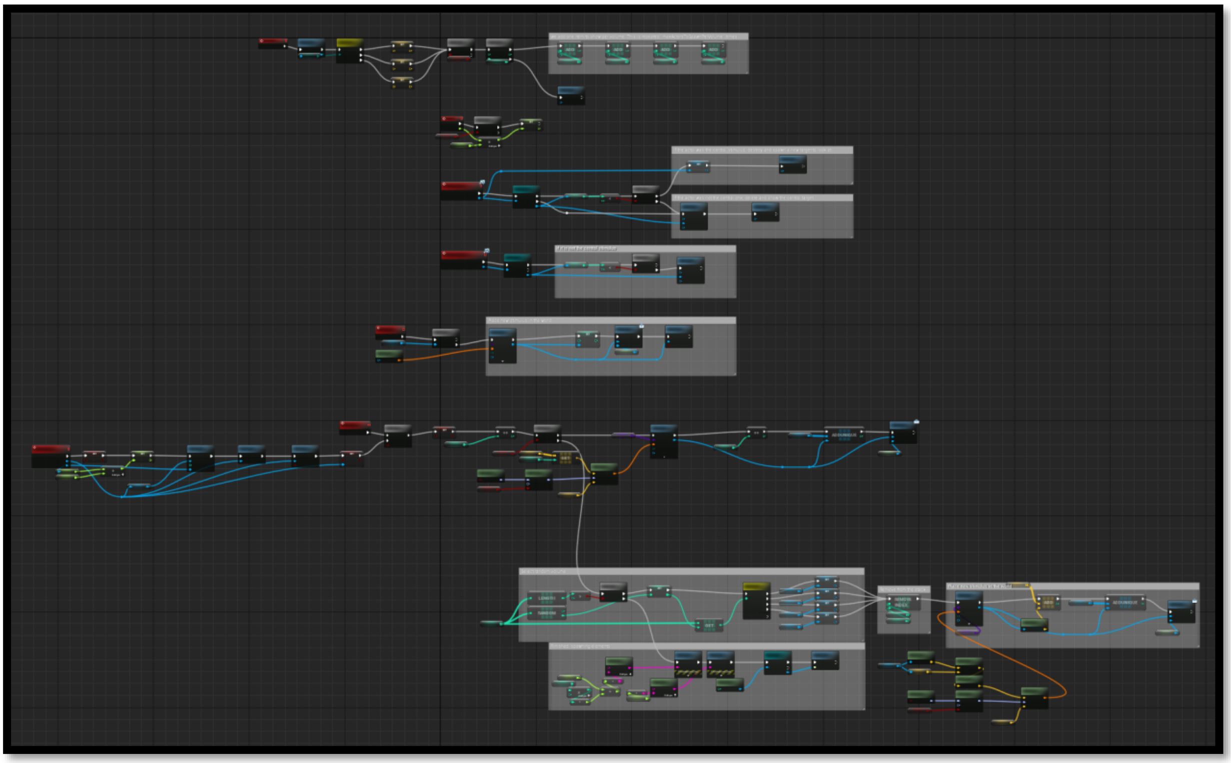



World Outliner

Search...

Label	Type
Saccades_B1_Lv (Editor)	World
Background_Bp	Edit Background_Bp
DirectionalLight	DirectionalLight
Fish_Bp	Edit Fish_Bp
Fish_Bp2	Edit Fish_Bp
Pond_FBX	StaticMeshActor
PostProcessVolume	PostProcessVolume
Saccade_Pwn	Edit Saccade_Pwn
Saccades_B_DataCollector_Bp	Edit Saccades_B_Data(
SkyLight	SkyLight
SphereReflectionCapture	SphereReflectionCapture
SphereReflectionCapture2	SphereReflectionCapture

Co-funded by the
Erasmus+ Programme
of the European Union



Settings

User and system settings are stored separately. Their interface and respective values being saved can be found in **`Core/Data`**.

All the calibration related settings, whether to track left, right or both pupils and other settings which might be helpful for people with MDVI can be set via the eye trackers' application - i.e. Tobii Experience and GazePoint Control. Details on this can be found in the user documentation.

Testing and Debugging

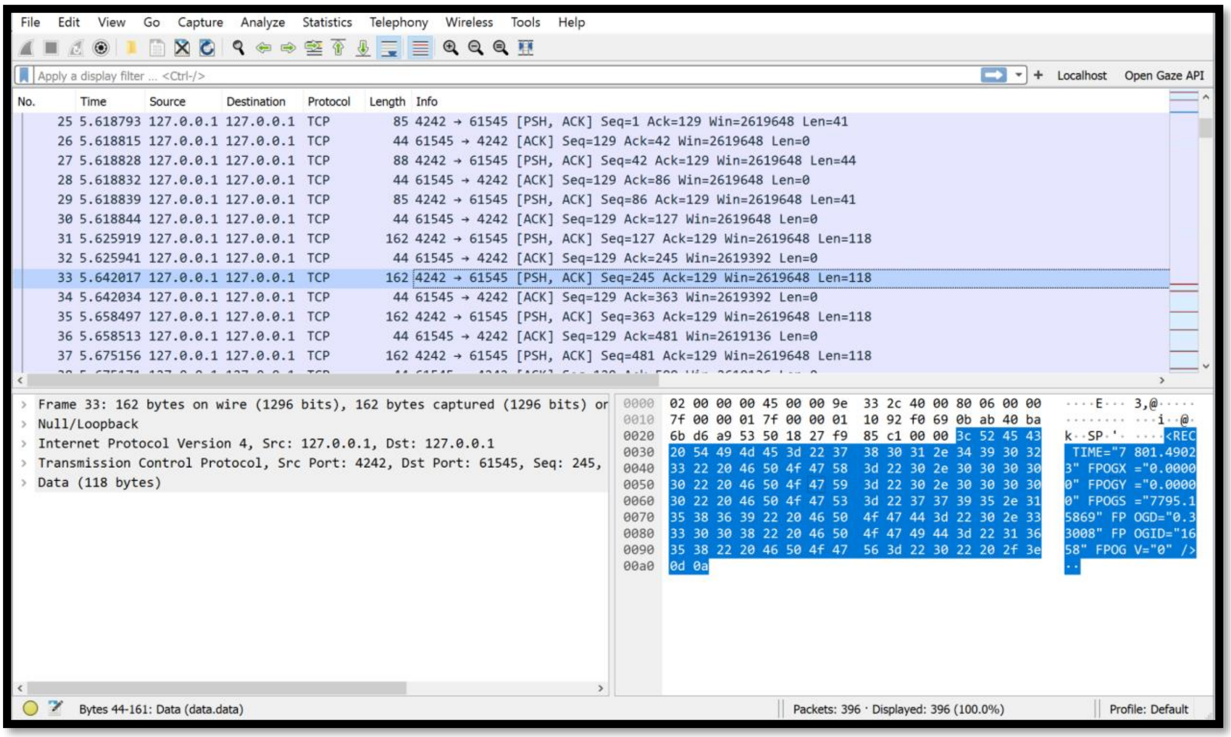
It's crucial to test and debug the application throughout the development process. Unreal Engine 4 provides tools and features to assist with debugging and profiling, allowing developers to identify and fix issues quickly and efficiently.

1. Unreal Engine Debugger: The built-in debugger in Unreal Engine 4 enables developers to step through Blueprints and examine variables during execution. This allows for close examination of how the code works, and helps to identify any logic errors or unexpected behaviors.
2. Unreal Engine Profiler: The profiler is a tool that helps developers measure performance and identify performance bottlenecks. It provides a detailed breakdown of the time spent on various tasks, such as rendering, physics, and

Blueprint execution. By examining the results, developers can optimize the application and improve its performance.

3. Debugging Eye Tracking: To debug eye tracking within the application, you can use the built-in Print String node in Blueprints to print the gaze data to the screen or log. This can help you visualize the eye tracking data in real-time and identify any issues or inconsistencies.
4. Debugging OpenGaze API: For debugging the communication with the OpenGaze API, you can use the Print String node in Blueprints to print the received XML strings, parsed values, or any other relevant information to the screen or log. This will help you identify any issues in the communication process and ensure that the data is being parsed and used correctly. More importantly tools like WireShark can help you snoop the traffic between the application and the API to detect potential issues and get a clearer insight into the messages being sent. Set the filter to `ip.dst === 127.0.0.1 && (tcp.dstport === 4242 || tcp.srcport === 4242)` to only view the relevant traffic.

Apart from these tools it is important to keep in mind that the eye tracker that is being used to test should be well calibrated. To simplify the testing and debugging process we implemented fallback support for the mouse as input device. Mouse input will be used when no supported eye tracker is detected.



```
No.  Time     Source    Destination Protocol Length Info
  25 5.618793 127.0.0.1 127.0.0.1   TCP       85 4242 → 61545 [PSH, ACK] Seq=1 Ack=129 Win=2619648 Len=41
  26 5.618815 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=42 Win=2619648 Len=0
  27 5.618828 127.0.0.1 127.0.0.1   TCP       88 4242 → 61545 [PSH, ACK] Seq=42 Ack=129 Win=2619648 Len=44
  28 5.618832 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=86 Win=2619648 Len=0
  29 5.618839 127.0.0.1 127.0.0.1   TCP       85 4242 → 61545 [PSH, ACK] Seq=86 Ack=129 Win=2619648 Len=41
  30 5.618844 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=127 Win=2619648 Len=0
  31 5.625919 127.0.0.1 127.0.0.1   TCP      162 4242 → 61545 [PSH, ACK] Seq=127 Ack=129 Win=2619648 Len=118
  32 5.625941 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=245 Win=2619392 Len=0
  33 5.642017 127.0.0.1 127.0.0.1   TCP      162 4242 → 61545 [PSH, ACK] Seq=245 Ack=129 Win=2619648 Len=118
  34 5.642034 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=363 Win=2619392 Len=0
  35 5.658497 127.0.0.1 127.0.0.1   TCP      162 4242 → 61545 [PSH, ACK] Seq=363 Ack=129 Win=2619648 Len=118
  36 5.658513 127.0.0.1 127.0.0.1   TCP       44 61545 → 4242 [ACK] Seq=129 Ack=481 Win=2619136 Len=0
  37 5.675156 127.0.0.1 127.0.0.1   TCP      162 4242 → 61545 [PSH, ACK] Seq=481 Ack=129 Win=2619648 Len=118
```

```
> Frame 33: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) or
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 4242, Dst Port: 61545, Seq: 245,
> Data (118 bytes)
```

```
0000  02 00 00 00 45 00 00 9e  33 2c 40 00 80 06 00 00   ····E··· 3,@·····
0010  7f 00 00 01 7f 00 00 01  10 92 f0 69 0b ab 40 ba   ········ ···i··@·
0020  6b d6 a9 53 50 18 27 f9  85 c1 00 00 3c 52 45 43   k··SP·'· ····<REC
0030  20 54 49 4d 45 3d 22 37  38 30 31 2e 34 39 30 32    TIME="7 801.4902
0040  33 22 20 46 50 4f 47 58  3d 22 30 2e 30 30 30 30   3" FPOGX ="0.0000
0050  30 22 20 46 50 4f 47 59  3d 22 30 2e 30 30 30 30   0" FPOGY ="0.0000
0060  30 22 20 46 50 4f 47 53  3d 22 37 37 39 35 2e 31   0" FPOGS ="7795.1
0070  35 38 36 39 22 20 46 50  4f 47 44 3d 22 30 2e 33   5869" FP OGD="0.3
0080  33 30 30 38 22 20 46 50  4f 47 49 44 3d 22 31 36   3008" FP OGID="16
0090  35 38 22 20 46 50 4f 47  56 3d 22 30 22 20 2f 3e   58" FPOG V="0" />
00a0  0d 0a                                              ··
```

Bytes 44-161: Data (data.data)

Packets: 396 · Displayed: 396 (100.0%)

Profile: Default

Deployment

Once the development and testing process is complete, it's time to deploy the application. Although Unreal Engine 4 offers a variety of deployment options, including Windows, Mac, Linux, console platforms, and more, we only confirmed that the eye trackers work with Windows. To deploy the OPEYE 2 application, follow these steps:

1. In the Unreal Editor, select "`File`" from the top menu, and then choose "`Package Project.`"
2. Choose the target platform you wish to deploy to (e.g., Windows, Mac, Linux).
3. Select the output directory for the packaged project.
4. The packaging process will begin, and once completed, you will find the executable and necessary files in the specified output directory.

After deployment, the OPEYE 2 application is ready to be distributed and used by end-users. Users will need to have the necessary eye-tracking hardware and software set up and calibrated for the application to function correctly - details on this can be found in the user documentation.

Co-funded by the
Erasmus+ Programme
of the European Union

Conclusion

This technical documentation has provided an overview of the OPEYE 2 project, including its setup, software architecture, eye-tracking implementation, and deployment process. Developers should now have a clear understanding of the structure and functionality of the project, as well as the methods and technologies utilized throughout its development.

With this knowledge, developers should be well-equipped to extend the project by implementing support for additional eye-tracking devices. To do so, they can study the existing integration of Tobii EyeTracker SDK and OpenGaze API, and follow similar patterns to incorporate other eye-tracking solutions into the project.

Furthermore, developers should have a firm grasp on how to create and add new levels to the OPEYE 2 application. By examining the existing levels and their associated Blueprint files, they can design and develop new levels with unique functionality, enabling the continued expansion and improvement of the application.

In summary, this documentation aims to equip developers with the necessary understanding of the OPEYE 2 project to facilitate future development, expansion, and maintenance.

Co-funded by the Erasmus+ Programme of the European Union