

Allen institute

Game controller/space navigator integration with Vaa3D/Mozak

# Contents

[1 Contents 1](#_Toc17465423)

[2 Background 2](#_Toc17465424)

[2.1 Vaa3D 2](#_Toc17465425)

[2.2 Game controllers 2](#_Toc17465426)

[2.3 The space navigator 3](#_Toc17465427)

[3 Building Vaa3D 4](#_Toc17465428)

[4 Integrating a game controller with Vaa3D 4](#_Toc17465429)

[4.1 Integration using the Game Controller API 4](#_Toc17465430)

[4.1.1 Setting up the game controller object in Vaa3D 4](#_Toc17465431)

[4.2 Integration using DS4Windows 6](#_Toc17465432)

[5 Using DS4Windows and embedded code together 7](#_Toc17465433)

# Background

Integration of a game controller and the “space-mouse” with Vaa3D/Mozak came out of a request from the neuron tracing group; to provide an alternative to using a regular mouse and keyboard. This could potentially help reduce repetitive motions leading to fatigue and other problems.

## Vaa3D

Vaa3D/Mozak is an application (written in C++) that compiles and builds using a development framework called QT. QT provide a programmer with numerous API’s, allows cross platform development (Mac, Windows, Linux) and provide the developer with visual tools helping designing and implementing graphical user interfaces (GUI’s), as well as debugging etc. It compiles on Windows using Visual Studio (or Clang). Current version of QT is 5.13**,** released on June 19 2019.

Vaa3D can only be compiled using Visual Studio 2013 and QT 4.8.6. QT 4.8.6 was released on Dec 15 2011. It is unsupported since 2015.

## Game controllers

A ‘regular’ game controller is a piece of hardware that has a set of buttons (12 or more on a regular PS4 controller for example), a Point of View (POV) control, and two joysticks. In addition, the controller may have a mouse pad and hardware functions for feedback, e.g. vibrating and colored LED’s. Applications that integrate such hardware need to be able to receive and send data to and from the controller. This is achieved using generalized (or specialized) API’s. The image below shows a PlayStation Dual shock controller.



Figure : A PlayStation Dual Shock controller

## The space navigator

The space mouse is a six-axis controller, with 2 buttons. The controller can be used to control the orientation and position of a 3D object in a volume, and is used as such in CAD applications for example.



Figure : The Space Navigator

# Building Vaa3D

As mentioned, Vaa3D can only be built using Visual Studio 2013 and QT version 4.8.6. Instructions for a Vaa3D build and all other required packages can be found here:

[https://github.com/Vaa3D/Vaa3D\_Wiki/wiki/Build-Vaa3D-on-Windows-with-qmake-using-VS2013-and-Qt4.8.6](https://nam05.safelinks.protection.outlook.com/?url=https%3A%2F%2Fgithub.com%2FVaa3D%2FVaa3D_Wiki%2Fwiki%2FBuild-Vaa3D-on-Windows-with-qmake-using-VS2013-and-Qt4.8.6&data=02%7C01%7C%7Cb5ffea6b28954c49acca08d6eebc0be0%7C32669cd6737f4b398bddd6951120d3fc%7C0%7C0%7C636958888758880209&sdata=n4uPRe5v42kwOoM4%2Fe3hsRyvwAKG6DY%2FuRjn9i078L4%3D&reserved=0)

The main Vaa3D code repository is here:

<https://github.com/Vaa3D/v3d_external>

A fork of the main repository is here:

<https://github.com/TotteKarlsson/v3d_external>

# Integrating a game controller with Vaa3D

Vaa3D is designed around QTPlugins and QTWidgets. Initially, integration of Game Controller code was to be done using the plugin framework. However, since any interaction with the UI must be done in the main thread, i.e. the UI thread, it seemed better to put the code directly in one of the widgets receiving UI events. The chosen unit is called *MozakUI.*

## Integration using the Game Controller API

An API allowing a client to easily assign custom actions for any Game Controller event was designed and implemented, partly based on previous code, see code repository:

<https://github.com/TotteKarlsson/ArrayBot>

However, that code was using a polling loop that can’t easily be used inside the QT framework. Instead a new subclass was created, *GameControllerRaw*. The ‘new’ game controller code got its own repository here:

<https://github.com/AllenInstitute/gamecontrollerLib>

The GameController raw class reads and translates Microsoft windows messages using the *Raw Input* protocol, see:

<https://docs.microsoft.com/en-us/windows/win32/inputdev/about-raw-input>

The Game Controller API is built using CMake, and depends on Dune Scientific’s dslFoundation library:

<https://github.com/TotteKarlsson/dsl>

### Setting up the game controller object in Vaa3D

The code integrating game controller functions in Mozak is placed in the MozakUI.h file (Figure 3).

As can be seen in Figure. 3, the mGC member is a unique pointer to a GameControllerRaw object.

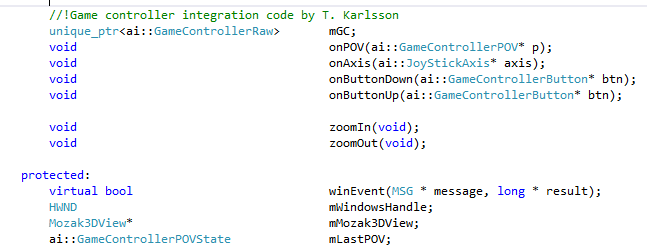


Figure : Game controller code in the MozakUI unit

The onPOV(), onAxis(), onButtonDown() etc. are member (callback) functions that are called in the main UI thread when a windows message is received as a result from user interaction with the game controller.

The winEvent() function is an over ridden system event providing function, allowing a client to handle a system event before the QT framework handles it.

In our case, there is only one event that is checked for, namely WM\_INPUT, which is the message delivering RawInput data (Figure 4).

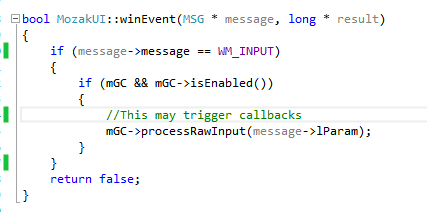


Figure : The winEvent function only dispatches   
“Raw Input” data to the gamecontroller object.

As seen in Figure 4, data is submitted to the Game Controller object (mGC), which will handle the message and call any user defined callbacks, e.g. onAxis(), onButtonDown() etc.

Game controller callbacks are assigned in the MozakUI constructor, e.g.:

…

mGC->mPOV.assignEvent(bind(&MozakUI::onPOV, this, \_1));

mGC->mJoyStick1.mXAxis.assignEvent(bind(&MozakUI::onAxis, this, \_1));

…

## Integration using DS4Windows

To be able to use a PlayStation game controller in Windows, 3rd party code is required.

The DS4Windows application see (<http://ds4windows.com/>), can detect and use a PlayStation controller.

The DS4Windows application runs robustly and expose the controller to windows as being a generic *GamePad* device. In fact, DS4Windows exposes the device as an Xbox controller, and all buttons and joystick messages are accessible using the RawInput protocol for a programmer.

In addition, the DS4Windows does also provide a graphical user interface, allowing a user to hook up actions to hardware device/sensor on the controller, including its inbuilt gyro and mouse pad (not accessible through RawInput). It also has a macro language, allowing complex actions to be recorded and mapped to an action on the controller.

Figure below shows the “Edit Profile” window for DS4Windows.

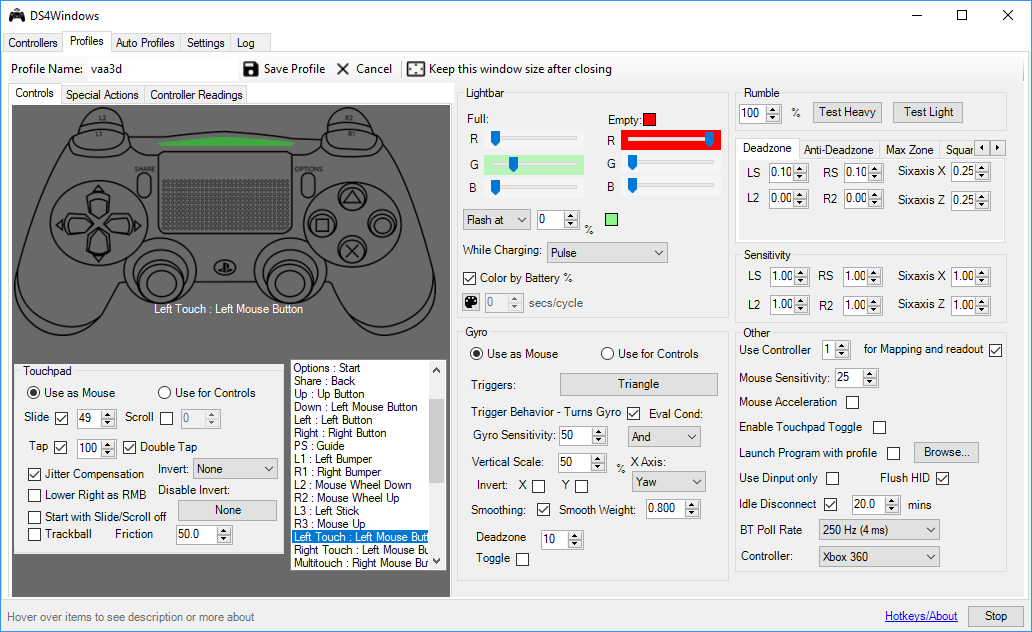


Figure : DS4Windows profile window.

The latest DS4Windows code and binaries (Version 1.7.15) are on GitHub:

<https://github.com/Ryochan7/DS4Windows/releases>

# Using DS4Windows and embedded code together

The ability to assign mouse/keyboard actions to the game controller by using the DS4Windows UI offers a quick and simple way of using a game controller together with Vaa3D/Mozak.

Embedded code, on the other hand, assures the user integrated functionality and requires no 3rd party software. However, depending on the actual functionality, the implementation cost can be high.

*So, an important question is, which functions should be assigned using the DS4Windows program, and which ones should be assigned using embedded code?*

In short, the embedded code and DS4Windows assignments need to be coordinated, as assigning a function on the controller using DS4Windows will cause the controller message arriving in Vaa3D’s message loop to be masked.

For example, if the triangle button is assigned in the embedded code to reset the 3D view, and at the same time, a DS4Windows action is assigned to the triangle button, then the embedded code will not receive a ‘triangle button is pressed’ message.

Assuming the assignment of the actions using DS4Windows and embedded code will initially be an evolving project, the table below are subject to change.

|  |  |  |
| --- | --- | --- |
| Action | Controller | Note |
|  | **Actions using embedded code** |  |
| Zoom in resolution | FrontRightTop Button |  |
| Zoom out resolution | FrontLeftTop Button |  |
| Zoom in view | FrontRightBottom Button |  |
| Zoom out view | FrontLeftBottom Button |  |
| Rotate view about X | POV West + second joystick |  |
| Rotate view about Y | POV North + second joystick |  |
| Rotate view about Z | POV East + second joystick |  |
|  |  |  |
|  | **Actions using DS4Windows** |  |
| Move mouse using gyro | Triangle button |  |
| Toggle Split | Square button |  |
| Toggle Virtual Finger | Cross button |  |
| Toggle Delete | Circle Button |  |
| Reset View | Press JoyStick #2 |  |

# Integrating a space-navigator device in mozak

In addition to the game controller integration discussed above, another device can be used with Mozak, namely a *SpaceNavigator* device. This device is composed of 6 individual axes that can be used for 3D translations and rotations.

# Other

The modified Vaa3D/Mozak application can be installed on a Windows computer using an installer from here:

[\\aibsfileprint\public\MozakReleases](file:///\\aibsfileprint\public\MozakReleases)

## Versioning

Code was added that checks for a version file. The current version will show in the MozakUI caption:

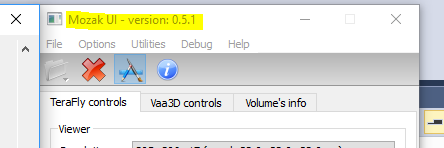


Figure : Figure showing a version number in the caption of the Mozak UI

## Version history

|  |  |  |
| --- | --- | --- |
| Version | Main Changes | Notes |
| 0.5.0 | Initial inclusion of game controller code into the Mozak UI unit. |  |
| 0.5.1 | Included code to handle messages from a SpaceNavigator device. |  |

## Logging

A log file is written to a user’s %USER%/AppData/Local/Vaa3D folder, named va3d-mozak.log.

This log file is overwritten every time the UI is started.