Pulse Unity Asset User Manual

v2.0 - https://goo.gl/GK1RZW

Table of contents

Resources

- Asset store page: here
- Discourse forum (support): here
- Issue tracker: here
- Pulse Physiology Engine documentation: here
- Pulse Unity Asset developer documentation: here

Overview

The role of this asset is to integrate the Pulse Physiology Engine in Unity. It includes the following directories:

- **Plugins**: Native and managed plugins to import part of the PulseEngine C++ API in Unity. See the developer documentation for more information on how to update them manually.
- **StreamingAssets**: Data files necessary for the Pulse engine to run and process actions. You will need to copy the contents of this inner directory to the StreamingAssets directory located in the Assets folder of your project.
- **Scripts**: Files implementing components and scriptable objects that can be used to conveniently interact with the Pulse Engine in your scene.
- **Demos**: Example data, scripts and scenes to showcase how to use the Pulse components.

If you need help designing a custom solution based on Pulse or want to discuss collaboration, please contact us at kitware@kitware.com.

Limitations

- **IL2CPP Build Support:** The Pulse Unity Asset is not supported in Unity projects built with the IL2CPP compiler. You can find more details in the developer documentation and receive support on our Discourse forum.
- **Missing functionality from the C++ framework:** The Pulse Unity Asset only exposes a portion of all the functionalities of the Pulse engine available in C++. The exhaustive list of functionality is defined in the Common Data Model. If you are interested in extending the Pulse Unity Asset to cover more of those functionalities, you can refer to the developer documentation and find support on our Discourse forum.

Unity components

While advanced users could rely solely on the C# API of the `PulseEngine` imported in the plugins, we have created some useful components meant to facilitate interacting with Pulse in Unity.

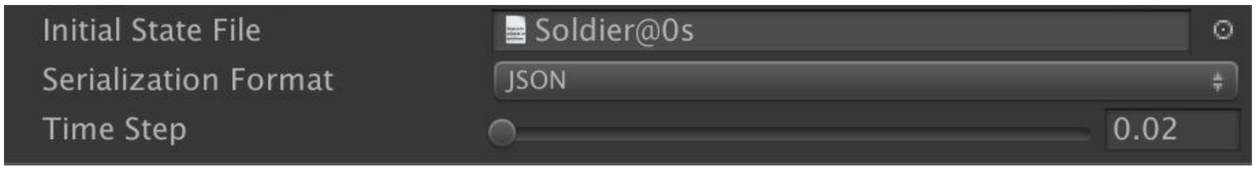
1. Generating Data

`PulseDataSource` is an abstract class which stores data organized per field in a `PulseData` scriptable object at each `MonoBehaviour.Update` and exposes it to be consumed by an instance of `PulseDataConsumer`. Its implemented subclasses are:

`PulseEngineDriver`

This is the main component for a Pulse simulation as it creates an instance of `PulseEngine` and handles advancing the simulation time to maintain synchronization with the game clock. It exposes the `PulseEngine` so that other components can apply actions to it (see below), and populates `PulseData` with the vitals information generated by the engine, including simulation time, ECG signal, heart rate, arterial, systolic, and diastolic blood pressures, oxygen saturation, end tidal carbon dioxide, respiration rate, temperature, airway carbon dioxide partial pressure, and blood volume.

Note*: This above list is only a subset of the outputs available in the C++ framework (learn more).*

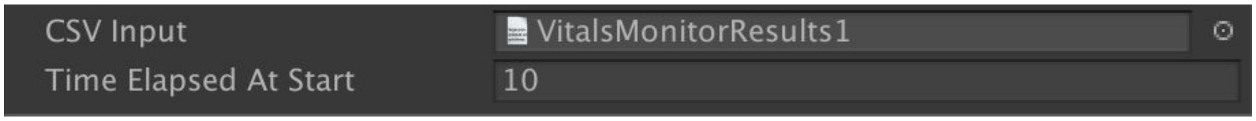


Initial State File: Soldier@0s
Serialization Format: JSON
Time Step: 0.02

- **Initial State File**: initial stable state to use to start the simulation. Example files can be found in *Demos > Data > states*. Because the physiology models require execution to achieve stable calculations, several stable patient states are provided.
- **Serialization Format**: serialization format of the state file (JSON or BINARY).
- **Time Step**: simulation time step in seconds.

`PulseCSVReader`

This component is used to display vital sign data stored in a comma separated value (csv) file from a previously executed pulse simulation.

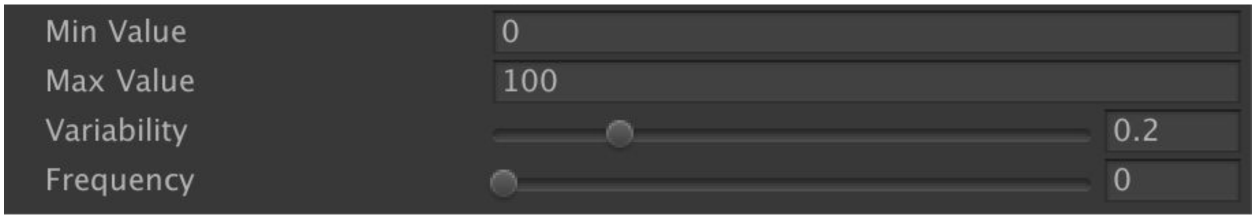


CSV Input: VitalsMonitorResults1
Time Elapsed At Start: 10

- **CSV Input**: input TextAsset containing Pulse output data.. An example file can be found in Demos > Data > csv.
- **Time Elapsed At Start**: offset time defining how many data points to read from the CSV file when the component is enabled.

`PulseRandomValueGenerator`

This is a convenient component used to generate random values in a `PulseData` container.

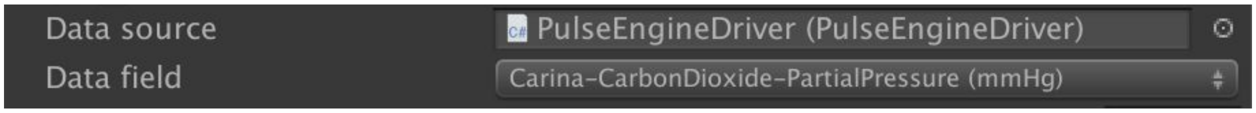


Min Value	0
Max Value	100
Variability	0.2
Frequency	0

- **Min/Max Value**: range of the values being generated.
- **Variability**: fraction defining how much different the next generated value will be.
- **Frequency**: frequency at which values are generated

2. Consuming vitals data

`PulseDataConsumer` is an abstract class that listens to an instance of a `PulseDataSource` to consume its `PulseData` for a certain data field at each `MonoBehaviour.LateUpdate`. Its subclasses (listed below) all inherit a custom editor with the following interface:

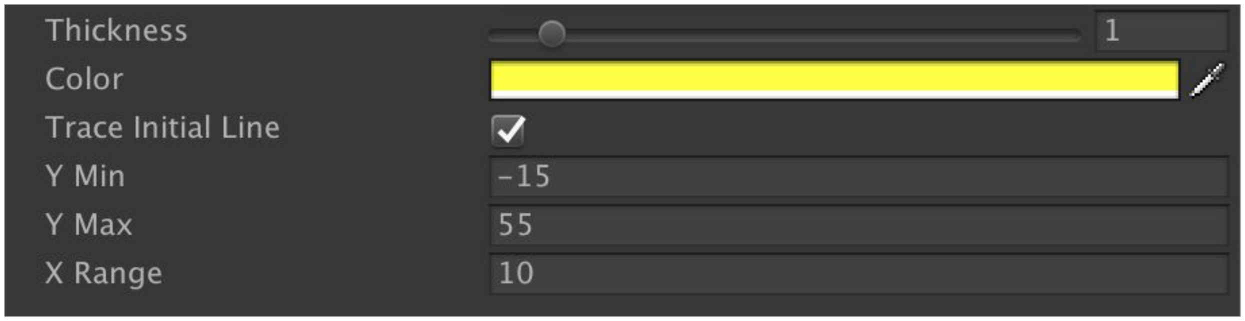


Data source	PulseEngineDriver (PulseEngineDriver)
Data field	Carina-CarbonDioxide-PartialPressure (mmHg)

- **Data source**: instance of the `PulseDataSource` to read the `PulseData` from.
- **Data field**: name of the data field to consume in the `PulseData` container.

`PulseDataLineRenderer`

Renders a line (embeds a `LineRenderer`) that draws points based on the data field selected. One of its parent needs a `UI.Canvas` component with a Render Mode set to "Screen Space - Camera" to overlay it on the screen, or to "World Space" to place it on a 2D plane in the 3D scene. Because the embedded `LineRenderer` is not part of the UI system, you can not set the canvas Render Mode to "Screen Space - Overlay".

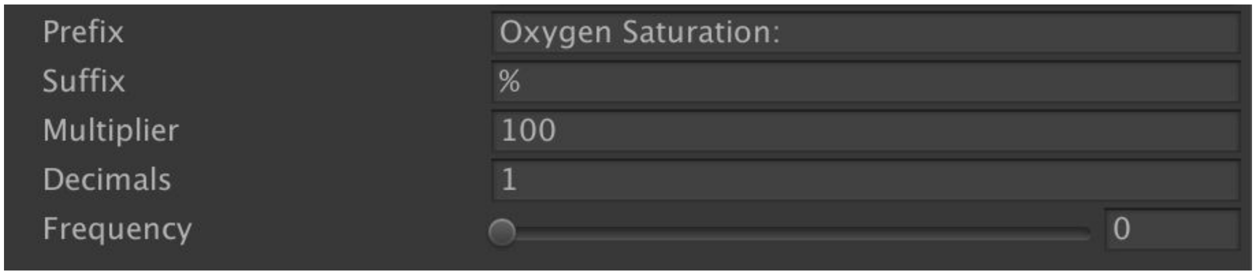


Thickness	1
Color	🟨
Trace Initial Line	✓
Y Min	-15
Y Max	55
X Range	10

- **Thickness:** thickness of the line renderer
- **Color**: color of the line renderer
- **Trace Initial Line**: shows a flat line at time points where there are no read values yet
- **Y Min/Max**: range the Y axis representing the values of the data field selected
- **X Range**: range of the X axis in seconds. This axis is dynamic, with the far right being "now" (latest value), and the far left being "now - X Range".

`PulseDataNumberRenderer`

Update the value of an associated UI.Text component to reflect the latest data field value. One of its parent needs a `UI.Canvas` component, and its Render Mode can be selected to World Space or Screen Space to respectively place it in the 3D world or overlay it on the screen.



Prefix	Oxygen Saturation:
Suffix	%
Multiplier	100
Decimals	1
Frequency	0

- **Prefix**: text prepended to the data value.
- **Suffix**: text appended to the data value.
- **Multiplier**: number multiplied to the value (useful to show percentages).
- **Decimals**: number of decimals shown.
- **Frequency**: frequency at which the number can be updated (0:no limit).

3. Interacting with the patient

With Pulse, you can **apply actions** on the simulated patient to introduce changes in the engine, either a traumatic event or a treatment step. Some of those actions involve **substances** circulating in the system, from drugs to gases.

In C#, the actions need to be passed to the `PulseEngine` object, which is exposed in the `PulseEngineDriver` component. To access it, you could elect to either:

- subclass the `PulseEngineController` class designed for this effect (which also enforces a custom editor), or
- simply add a reference to a `PulseEngineDriver` in your custom component that defines when to apply an action.

The exposed actions with their inputs are in the table. Examples of how they can be used are available in the `PulseActionOnClick` example class, located inside *Demos > Scripts* . The **available substances** are listed in *StreamingAssets > PulseDataFiles > substances*, using the same string as the substance file name.

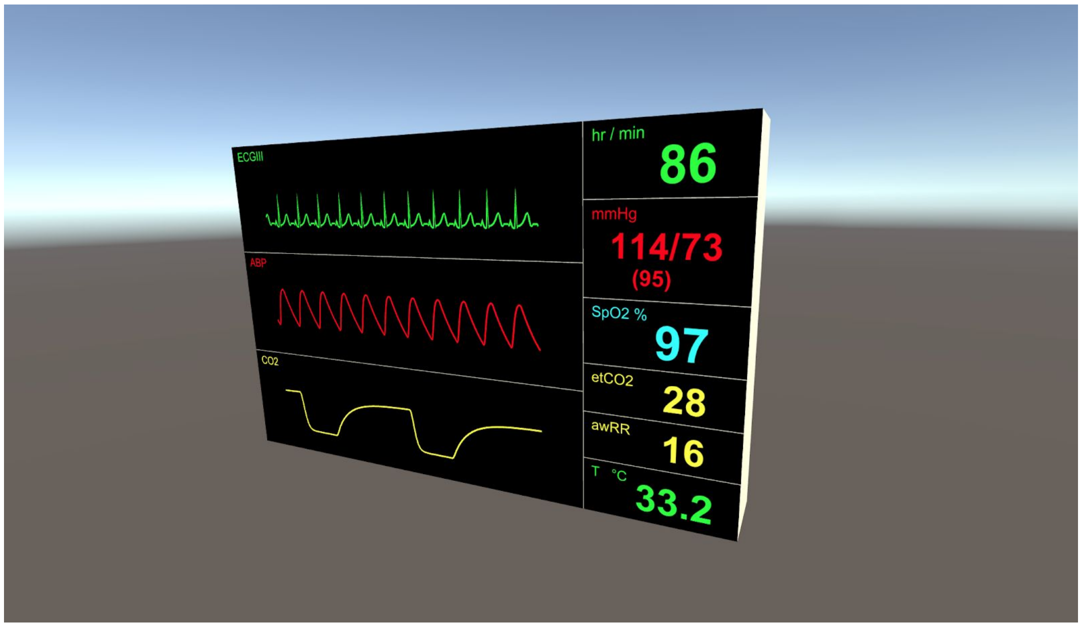
Action	**Input**
Hemorrhage	Rate, Type (Internal/External), Compartment (ex. Vena Cava, Right Leg, Aorta, Spleen)
SubstanceCompoundInfusion	Substance (ex. Saline), Bag Volume, Rate
AirwayObstruction	Severity (0 to 1)
Intubation	Type (ex. Tracheal, Off)
TensionPneumothorax	Side (Right/Left), Type (Open/Closed), Severity (0 to 1)
NeedleDecompression	Side (Right/Left), State (On/Off)
SubstanceBolus	Substance, Concentration, Rate
AnesthesiaMachineConfiguration	Connection (ex. Tube, mask), InletFlow, InspiratoryExpiratoryRatio, OxygenFraction, OxygenSource (ex. wall), PrimaryGas, RespiratoryRate, VentilatorPressure

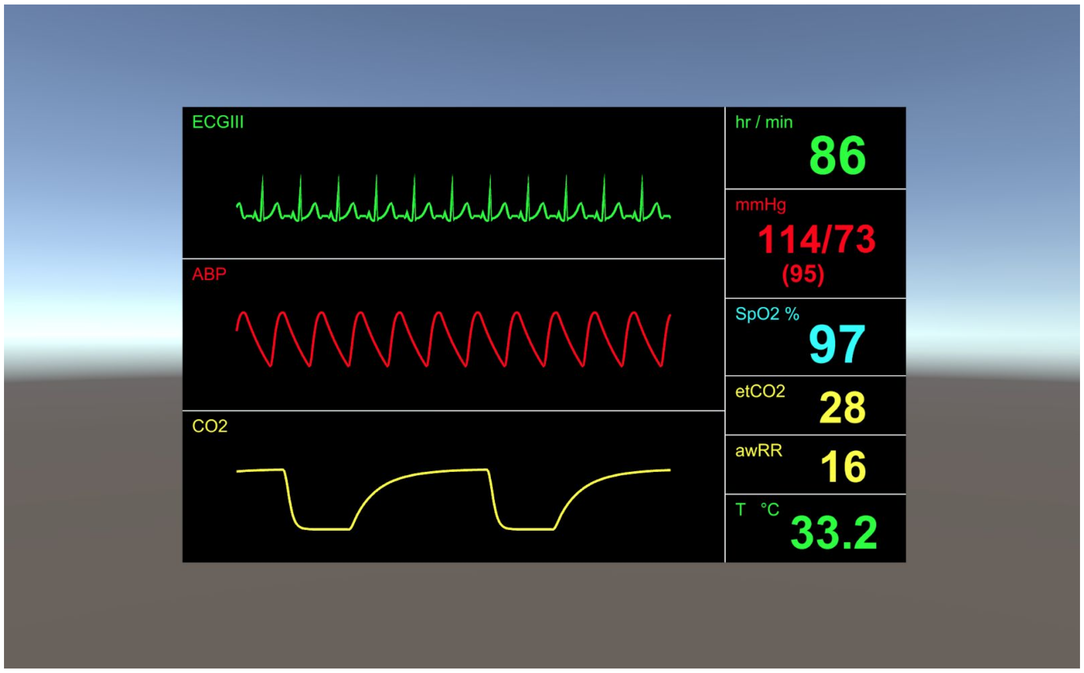
***Note**: those are only a subset all the actions available in the C++ framework (learn more).*

Examples

VitalsMonitor

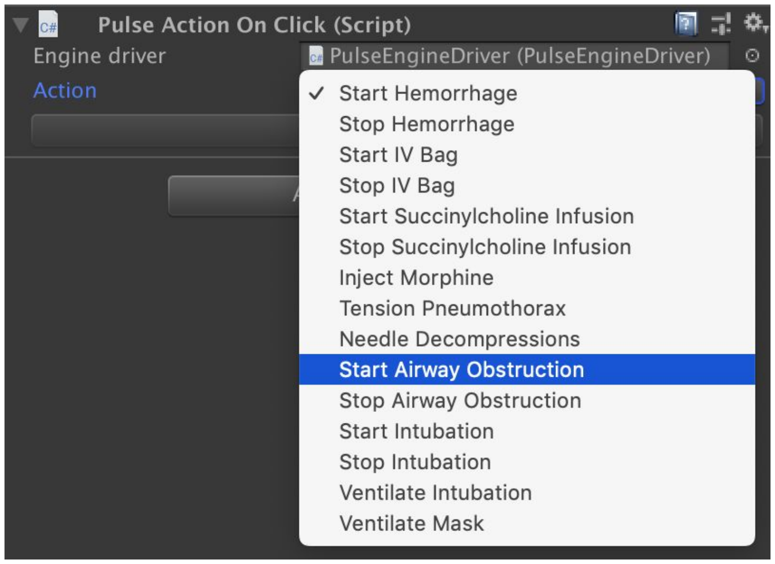
This example scene showcases a full vitals monitor screen similar to the Pulse Explorer, constructed with multiple `PulseDataLineRenderer` and `PulseDataNumberRenderer`. Its input data is generated by a `PulseEngine` through the `PulseEngineDriver` component.

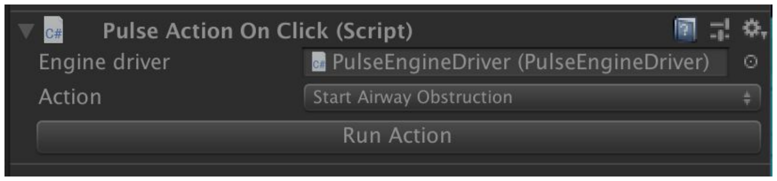




CreateActionOnClick

This scene showcases a simple canvas in world space with a single `PulseDataLineRenderer` and `PulseDataNumberRenderer`, using data exposed through a `PulseEngineDriver`. The `PulseActionOnClick` component (implemented in Demos/Scripts) is added into the scene to be able to apply an action from a preset list by pressing the “Run Action” button when the simulation is running. These examples provide example uses of each exposed action and inputs that can be used to simulate patient injury and treatment.



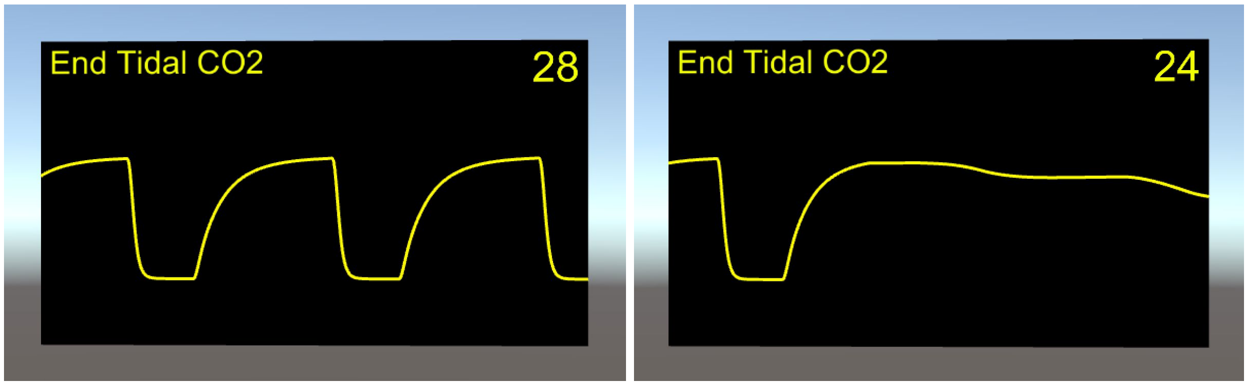


Pulse Action On Click (Script)

Engine driver: PulseEngineDriver (PulseEngineDriver)

Action: Start Airway Obstruction

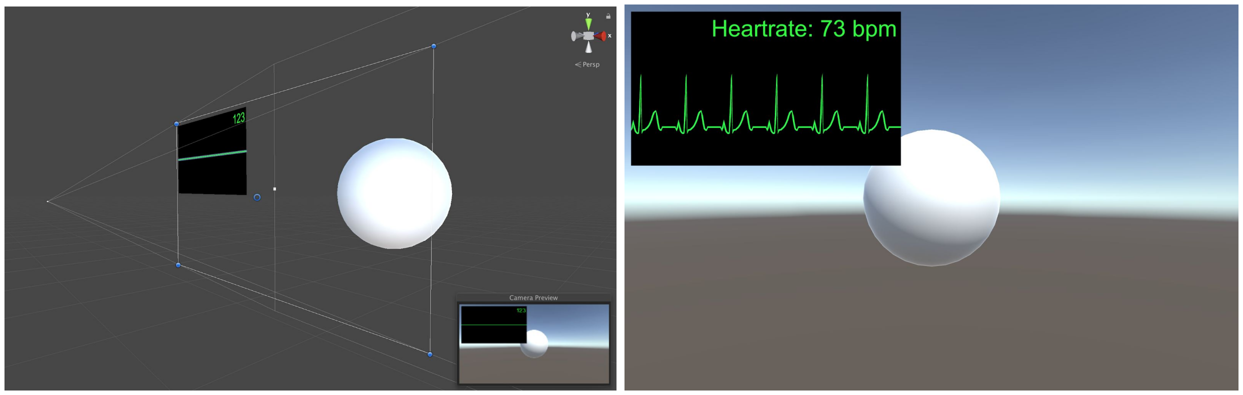
Top: Selecting an action in the PulseActionOnClick editor;
Bottom: PulseActionOnClick editor with enabled “Run Action” button when running.



Left: Capnogram and End-Tidal CO2 of an healthy patient;
Right: Same vital information after applying an airway obstruction.

ScreenSpace

This scene showcases a canvas in screen space (overlay) with a `PulseDataLineRenderer` and a `PulseDataNumberRenderer`, using data exposed through a `PulseCSVReader`.



Left: scene view with canvas overlayed on the camera plane; Right: game view.

RandomNumber

This scene showcases a simple canvas in world space with a single `PulseDataLineRenderer` and `PulseDataNumberRenderer`, using data generated with a `PulseRandomValueGenerator`.