Visualizer Studio

for Unity 3d

by Altered Reality Entertainment

v1.3.0

Overview

General Description

Visualizer Studio is a Unity Scripting Package which enables the developer to allow their game to react to music and sound effects. It exposes easy to use, powerful, and expandable base classes and interfaces that perform a lot of the difficult tasks of audio visualization. Using Visualizer Studio allows your game to create a range of effects, from subtly swaying trees to the music to driving all gameplay based off of the music.

Use Cases

- Some use cases of Visualizer Studio are as follows...
- Controlling the players based on the music.
- Spawning particle effects on beat..
- Creating a fully visualized background based on game music and sound effects.
- Spawn prefabs on beat..
- Making physics objects bounce along with the music.
- Creating a level for a platformer that reacts and moves with the music.
- Scripting enemies to attack in sync with the music.
- And many more!

System Overview

Visualizer Studio was created to encapsulate as much of the work with visualizing music and sound as possible, with still leaving it completely open to customization. The general overview of how Visualizer Studio works is as follows:

- The ***Manager*** is the object that interacts direction with the music and/or audio stream. It actually performs the data transformation that drives Visualization.
- The ***Data Group*** interacts directly with the Manager, and is setup to analyze a specific subset of the audio spectrum.
- A ***Controller*** interacts either with the Data Group, or on its own if it is not driven by the music. A Controller is the main object that the game will interact with when using Visualizer Studio.
- A ***Modifier*** interacts directly with a Controller and will modify Game Objects (or anything for that modifier) each frame along with changes in the music. This is used to encapsulate common visualization functions, instead of always interacting directly with a Controller
- A ***Trigger*** interacts directly with a Controller and will modify Game Objects (or anything for that modifier) at peaks in the music. This is used to encapsulate common visualization functions, instead of always interacting directly with a Controller.

Basic Usage Walkthrough

1. Create or Open a Scene

2. Add the included Manager Prefab

a) Add the prefab, VisualizerManager, to your scene.

b) Everything should now be setup for a basic usage of a Manager and Data Groups covering the Bass, Mid, and High audio frequencies.

c) Everything should now be setup for a basic usage of Frequency Controllers covering the Bass, Mid, and High audio frequencies.

3. Add Modifiers and Triggers to Game Objects

a) Find any Game Objects that you desire to react to the music.

b) Add either a Trigger or Modifier to the Game Object from "Component=>Visualizer Studio=>Modifiers" or "Component=>Visualizer Studio=>Triggers"

c) Select the previously created Manager in the Inspector for this Modifier/Trigger.

d) Select the desired Controller in the Inspector that this Modifier/Trigger should connect to.

e) Configure the new Trigger/Modifier through the Inspector to make your Game Object react to the music as you desire.

For advanced usage, please refer to the section at the end of this document.

<u>High Level Scripts</u>

VisManager

This is the main class that deals with managing all of the main visualizer spectrum data. All visualizer objects eventually interact with this class. This class should not be overridded.

Inspector properties:

- Channel
    - This is the audio channel to look at for visualization.
- Window Size
    - This is the size of the spectrum window to use when visualizing the audio data. Higher window sizes give more accuracy at the cost of reduced performance. Smaller window sizes give greater performance, but lower accuracy.
- Window Type
    - This indicates what type of FFTWindow to use when creating the spectrum visualization data. Refer to the following Unity support page for more information on the FFTWindow: http://unity3d.com/support/documentation/ScriptReference/FFTWindow.html

- Audio Source
  - This is the AudioSource to use for creating the visualization data. This should be set in order for the Manager to work properly. Otherwise, it will use the static Audio Listener, which listens to ALL of the audio in the game. This will not work as well as connecting directly to an Audio Listener.

VisDataGroup

This class is used to encapsulate subsections of the visualization spectrum data. This is specifically used to isolate certain frequency ranges; such as Bass, Mid, and High frequencies. This class should not be overridden.

Inspector properties:

- Data Group Name
  - This is the name of the data group that will be used to indentify it when selecting data groups from frequency controllers.
- Data Source
  - This indicates what data source of the visualization data this Data Group should look at. Spectrum indicates the transformed audio data, which is almost always what should be used. Raw Audio indicates the raw audio data of the input audio stream.
- Number Sub Data Groups
  - This is the number of sub data groups to split this Data Group into. This is used for increasing the accuracy of how data is aggregated for this data group.
- Frequency Range Start Index
  - This is the index of the spectrum data that this Data Group should start at.
- Frequency Range End Index
  - This is the index of the spectrum data that this Data Group should end at.
- Boost

    - This is how much to boost the data values aggregated by this Data Group. In other words, all data values will be multiplied by this value in order to boost the data values.

- Cutoff
    - This is the maximum value that any data value aggregated by this Data Group can have. In other words, all values will be cut off at this amount, and cannot be greater than this value.

VisBaseController

This class is the base class for all controllers. It provides all of the base functionality for manipulating a controller and changing it's value.

Inspector properties:

- Controllller Name
    - This is the name used to identify this Controller when selecting Controllers through the inspector.
- Limit Increase Rate
    - This indicates if the rate at which the controller value increases should be limited, or if it should increase instantly when increasing.
- Increase Rate
    - This is the maximum rate at which the controller value will be allowed to increase per second if the increase rate is currently being limited.
- Limit Decrease Rate
    - This indicates if the rate at which the controller value decreases should be limited, or if it should decrease instantly when decreasing.
- Increase Rate
    - This is the maximum rate at which the controller value will be allowed to decrease per second if the decrease rate is currently being limited.

VisBaseModifier

This is the base class for all modifiers. It provides all of the helper functions and base functionality to modify values each frame, based on the music. This class should be derived from, and not used directly.

N**o inspector properties in base class**

VisBaseTrigger

This is the base class for all triggers. It provides all of the helper functions and base functionality to modify values , or trigger actions, in reaction to peaks in the music. This class should be derived from, and not used directly.

Inspector properties:

- Trigger Type
  - This describes the type of this trigger. (i.e. How it functions) The types of functionality are as follows:
    - None
      - This indicates that this trigger will not be triggered automatically and must be triggered through custom functionality.
    - LessThanValueThreshold
      - This indicates that this trigger is triggered when the associated controllers value goes under the specified threshold.
    - GreaterThanValueThreshold
      - This indicates that this trigger is triggered when the associated controllers value goes over the specified threshold.
    - LessThanChangeThreshold
      - This indicates that this trigger is triggered when the associated controllers value difference for a specific frame goes under the specified threshold.
    - GreaterThanChangeThreshold
      - This indicates that this trigger is triggered when the

associated controllers value difference for a specific frame goes over the specified threshold. **This is the most common value for Trigger Type.**

- Trigger Threshold
  - This is the threshold at which the trigger should activate, based on the current Trigger Type.
- Trigger Reactivate Delay
  - This is the amount of time to wait in between consecutive triggers of this trigger.
- Trigger Random Reactivate Delay
  - This is the amount of random delay to add to the reactivate time. Random range from "-value" to "value".

VisBasePropertyModifier

This class is used as a base class for any modifier that is going to be setting basic properties on an object. For example, setting the scale of an object.

Inspector properties:

- Controller Source Value
  - This indicates what source value from the associated controller should be used to set the target property. The source value types are as follows:
    - Current
      - This indicates the current value of the associated controller.
    - Previous
      - This indicates the previous value of the associated controller.
    - Difference
      - This indicates the difference between the current and previous value of the associated controller.
- Min Controller Value

    - This is the min contoller value that should be looked at to determine how to modify the target property.

- Max Controller Value
    - This is the max controller value that should be looked at to determine how to modify the target property.
- Min Property Value
    - This is the min value that the target property should be set to.
- Max Property Value
    - This is the max value that the target property should be set to.
- Invert Value
    - This indicates if the target property should be modifier in reverse. For example, it starts are maxPropertyValue when the controller is at minControllerValue, and then moves to minPropertyValue when the maxControllerValue is set on the controller.

VisBasePropertyTrigger

This class is used as a base class for any trigger that is going to be setting basic properties on an object. For example, setting the scale of an object.

Inspector properties:

- Controller Source Value
    - This indicates what source value from the associated controller should be used to set the target property. The source value types are as follows:
        - Current
            - This indicates the current value of the associated controller.
        - Previous
            - This indicates the previous value of the associated controller.
        - Difference

        - This indicates the difference between the current and previous value of the associated controller.

- Min Controller Value
    - This is the min contoller value that should be looked at to determine how to modify the target property.
- Max Controller Value
    - This is the max controller value that should be looked at to determine how to modify the target property.
- Min Property Value
    - This is the min value that the target property should be set to.
- Max Property Value
    - This is the max value that the target property should be set to.
- Invert Value
    - This indicates if the target property should be modifier in reverse. For example, it starts are maxPropertyValue when the controller is at minControllerValue, and then moves to minPropertyValue when the maxControllerValue is set on the controller.
- Random Value
    - This indicates if, when triggered, this property trigger should pick a random value in the range specified, or if it should try to map/convert the value based on the ranges specified.
- Limit Increase Rate
    - This indicates if the increase rate of the target property value should be limited
- Increase Rate
    - This is the rate at which the target property value can increase per second.
- Limit Decrease Rate
    - This indicates if the decrease rate of the target property value

  - should be limited.
- Decrease Rate
  - This is the rate at which the target property value can increase per second.
- Return To Rest
  - This indicates if, after this trigger reaches its target property value, it will return back to its "rest" value. This can only be enabled if either the increase or decrease rate is limited, as without it, it would be set instantly.
- Rest Value
  - This is the "rest" value the property value should return to after the current target property value is hit.

Included Controllers

VisFrequencyController

This controller is driven by being directly connected to a data group, allowing it to react to the music.

Inspector properties:

- Sub Value Type
  - This is the value type to pull from all of the sub data groups on the target data group. Such as the Maximum of each sub data group values.
- Final Value Type
  - This is the final value to pull from the result of the Sub Value Type property. Such as the Average (Final Value Type) of the Maximums (Sub Value Type).

VisRandomController

This is a non music based controller that will pick random values between 0.0 and 1.0 after set intervals of time.

Inspector properties:

- Delay Time
    - This is the amount time to wait between picking random values.

VisSineWaveController

This is a non music based controller that will use a sine wave function to oscillate between 0.0 and 1.0.

Inspector properties:

- Speed Scalar
    - This is the speed scalar that controls how fast this controller oscillates between 0.0 and 1.0. The higher the scalar, the faster the osicillation.

Creating New Controllers

A large variety of custom controllers can be created by deriving either from VisBaseController or VisFrequencyController. In order to create a custom controller, you must override the following functions:

- void Reset()
    - This function is called whenever your script needs to be reset in the editor. You MUST make sure to call “base.Reset()” as the first thing in this function.
- void Start()
    - This function is called whenever your script is started running in the game. You MUST make sure to call “base.Start()” as the first thing in this function.
- float GetCustomControllerValue()
    - This function is called by VisBaseController to get the current desired value for your custom controller. This is where you perform all custom controller functionality.
- OPTIONAL: Rect DisplayDebugGUI(...)
    - This is an optional function that can be overridden to provide custom

debug GUI support for your controller. You must return a Rect the size of the control that was drawn.

IMPORTANT: To have your new controller properly working in the inspector, you must create a custom editor for it. This is usually as simple as creating a new editor and deriving from VisBaseControllerEditor. Look at VisFrequencyControllerEditor.cs or VisRandomControllerEditor.cs for reference.

Included Modifiers

VisAnimationStatePropertyModifier

This modifier is used to set animation state properties, such as animation speed.

Inspector properties:

- Target Property
    - This is the animation state property that this modifier should be changing.
- Target Animation
    - This is the target animation to change the state properties for. This must be playing in order to modify its state.

VisEmitterPropertyModifier

This modifier is used to set emitter properties, such as Max Size.

Inspector properties:

- Target Property
    - This is the emitter property that this modifier should be changing.

VisGameObjectPropertyModifier

This modifier is used to set basic game object properties, such as Uniform Scale.

Inspector properties:

- Target Property
    - This is the basic game object property that this modifier should be changing.

VisLightPropertyModifier

This modifier is used to set light properties, such as Intensity.

Inspector properties:

- Target Property
    - This is the light property that this modifier should be changing.

VisMaterialLerpModifier

This modifier is used to lerp between two materials.

Inspector properties:

- Lerp From Material
    - This is the material to lerp from.
- Lerp To Material
    - This is the material to lerp to.

VisMaterialPropertyModifier

This modifier is used to set float properties on a material.

Inspector properties:

- Target Property
    - This is the string name of a float property on this material that should be modified.
- Set as Procedural Material
    - This indicates if this material property modifier should tread the target as a procedural material and attempt to set the target property as a procedural float.
- Rebuild Procedural Textures Immediately
    - This indicates if the procedural textures should immediately be regenerated when setting a float on the target material
- Apply to Material Index

  - This indicates if this material property modifier should apply to a material in a specific index, or the main material.
- Material Index
  - This is the material index to apply this material propery change to.

VisTargetModifier

This modifier is used to target specific game objects, and to notify any components within that implement IVisModifierTarget. This is useful for allowing a modifier to control and notify many game objects when it is updated.

Inspector properties:

- Target Game Objects
  - This list contains all of the game objects that should be notified when the modifier value has changed.

IVisModifierTarget

- *Description*
  - Any game objects that are added to the list of “Target Game Objects” for the VisTargetModifier object MUST implement IvisModifierTarget.
- Abstract Functions
  - void OnValueUpdated(...)
    - float current
      - The current value of the targeted controller.
    - float previous
      - The previous frame value of the targeted controller.
    - float difference
      - The value difference of the targeted controller, between current and previous value.
    - float adjustedDifference

- The adjusted value difference of the targeted controller. This value is the difference value as if it took place over a certain time period, controlled by VisBaseController.mc_fTargetAdjustedDifferenceTime. The default of this essentially indicates a frame rate of 60 fps to determine the adjusted difference. This should be used for almost all difference calculations, as it is NOT frame rate dependent.

Creating New Modifiers

A large variety of custom modifiers can be created by deriving either from VisBaseModifier or VisBasePropertyModifier. In order to create a custom modifier, you must override the following functions:

- void Reset()
  - This function is called whenever your script needs to be reset in the editor. You MUST make sure to call “base.Reset()” as the first thing in this function.
- void Start()
  - This function is called whenever your script is started running in the game. You MUST make sure to call “base.Start()” as the first thing in this function.
- **If deriving from VisBaseModifier:** void OnValueUpdated(...)
  - This function is called by the base modifier whenever the value of the targeted controller is updated. All custom modifier work should be done in this function, as a result of the values passed in.
  - Parameters:
    - current
      - This function is called by the base modifier whenever the value of the targeted controller is updated.
    - previous
      - The current value of the targeted controller.

        - difference
            - The value difference of the targeted controller.
        - adjustedDifference
            - The adjusted value difference of the targeted controller. This value is the difference value as if it took place over a certain time period, controlled by VisBaseController.mc_fTargetAdjustedDifferenceTime. The default of this essientially indicates a frame rate of 60 fps to determine the adjusted difference. This should be used for almost all difference calculations, as it is NOT frame rate dependent.

- **If deriving from VisBasePropertyModifier:** void SetProperty()
    - This function is called by the base property modifier in order for the derived class to set its target property to the new value specified.

IMPORTANT: To have your new modifer properly working in the inspector, you must create a custom editor for it. This is usually as simple as creating a new editor and deriving from VisBaseModifierEditor. Look at VisAnimationStatePropertyModifierEditor.cs or VisTargetModifierEditor.cs for reference.

Included Triggers

VisAddForceTrigger

This trigger is used to apply forces to physics objects as a reaction to changes in the music.

Inspector properties:

- Controller Source Value
    - This indicates what source value from the associated controller should be used to set the target property. The source value types are as follows:
        - Current
            - This indicates the current value of the associated controller.

        - Previous
            - This indicates the previous value of the associated controller.
        - Difference
            - This indicates the difference between the current and previous value of the associated controller.
- Min Controller Value
    - This is the min contoller value that should be looked at to determine how to modify the target property.
- Max Controller Value
    - This is the max controller value that should be looked at to determine how to modify the target property.
- Min Force Value
    - This is the min value of the force to add.
- Max Force Value
    - This is the max value of the force to add.
- Invert Value
    - This indicates if the target property should be modifier in reverse. For example, it starts are maxForceValue when the controller is at minControllerValue, and then moves to minForceValue when the maxControllerValue is set on the controller.
- Random Value
    - This indicates if, when triggered, this property trigger should pick a random value in the range specified, or if it should try to map/convert the value based on the ranges specified.
- Force Direction
    - This is the world direction in which to apply the force.
- Force Mode

    - This is the mode of the force to apply.

VisAnimationStatePropertyTrigger

This trigger is used to set an animation state property as a reaction to changes in the music.

Inspector properties:

- Target Property
    - This is the animation state property that this modifier should be changing.
- Target Animation
    - This is the target animation to set the animation state for. This must be playing in order to modify its state.

VisEmitParticlesTrigger

This trigger is used to emit particles as a reaction to changes in the music.

Inspector properties:

- None

VisEmitterPropertyTrigger

This trigger is used to set an emitter property as a reaction to changes in the music.

Inspector properties:

- Target Property
    - This is the emitter property that this modifier should be changing.

VisGameObjectPropertyTrigger

This trigger is used to set a game object property as a reaction to changes in the music.

Inspector properties:

- Target Property
    - This is the game object property that this modifier should be changing.

VisLightPropertyTrigger

This trigger is used to set a light property as a reaction to changes in the music.

Inspector properties:

- Target Property
    - This is the light property that this modifier should be changing.

VisMaterialLerpTrigger

This trigger is used to lerp between two materials as a reaction to changes in the music.

Inspector properties:

- Lerp from Material
    - This is the material to lerp from.
- Lerp to Material
    - This is the material to lerp to.

VisMaterialPropertyTrigger

This trigger is used to set a float material property as a reaction to changes in the music.

Inspector properties:

- Target Property
    - This is the string name of a float property on this material that should be modified.
- Set as Procedural Material
    - This indicates if this material property trigger should tread the target as a procedural material and attempt to set the target property as a procedural float.

- Rebuild Procedural Textures Immediately
    - This indicates if the procedural textures should immediately be regenerated when setting a float on the target material
- Apply to Material Index
    - This indicates if this material property modifier should apply to a material in a specific index, or the main material.
- Material Index
    - This is the material index to apply this material propery change to.

VisMessageTrigger

This trigger is used to target specific game objects, and then send a message to all target game objects when triggered.

Inspector properties:

- Message Name
    - This is the name of the message to send.
- Message Parameter
    - This is this parameter to send with the message.
- Target Game Objects
    - This is the list of game objects that should be targeted.

VisSpawnPrefabTrigger

This trigger is used to spawn a prefab as a reaction to changes in the music.

Inspector properties:

- Prefab
    - This is the prefab to spawn when this trigger is triggered.
- Random Offset
    - This is the random offset to apply to the position when spawning.

VisTargetTrigger

This trigger is used to target specific game objects, and to notify any components within that implement IVisTriggerTarget. This is useful for allowing a trigger to control and notify many game objects when it is triggered.

Inspector properties:

- Target Game Objects
    - This is the list of game objects that should be targeted.
- **IVisTriggerTarget**
    - *Description*
        - Any game objects that are added to the list of "Target Game Objects" for the VisTargetModifier object MUST implement IvisModifierTarget.
    - Abstract Functions
        - void OnTriggered(...)
            - float current
                - The current value of the targeted controller.
            - float previous
                - The previous frame value of the targeted controller.
            - float difference
                - The value difference of the targeted controller, between current and previous value.
            - float adjustedDifference
                - The adjusted value difference of the targeted controller. This value is the difference value as if it took place over a certain time period, controlled by VisBaseController.mc_fTargetAdjustedDifferenceTime. The default of this essentially indicates a frame rate of 60 fps to determine the adjusted difference. This should be used for almost all difference calculations, as it is NOT frame rate dependent.

Creating New Triggers

A large variety of custom triggers can be created by deriving either from VisBaseTrigger or VisBasePropertyTrigger. In order to create a custom trigger, you must override the following functions:

- void Reset()
  - This function is called whenever your script needs to be reset in the editor. You MUST make sure to call "base.Reset()" as the first thing in this function.
- void Start()
  - This function is called whenever your script is started running in the game. You MUST make sure to call "base.Start()" as the first thing in this function.
- **If deriving from VisBaseTrigger:** void OnTriggered(...)
  - This function is called by the base trigger whenever the trigger has been triggered. All custom trigger work should be done in this function, as a result of the values passed in.
  - Parameters:
    - current
      - This function is called by the base modifier whenever the value of the targeted controller is updated.
    - previous
      - The current value of the targeted controller.
    - difference
      - The value difference of the targeted controller.
    - adjustedDifference
      - The adjusted value difference of the targeted controller. This

value is the difference value as if it took place over a certain time period, controlled by VisBaseController.mc_fTargetAdjustedDifferenceTime. The default of this essientially indicates a frame rate of 60 fps to determine the adjusted difference. This should be used for almost all difference calculations, as it is NOT frame rate dependent.

- **If deriving from VisBasePropertyTrigger:** void SetProperty()
  - This function is called by the base property trigger in order for the derived class to set its target property to the new value specified.

IMPORTANT: To have your new trigger properly working in the inspector, you must create a custom editor for it. This is usually as simple as creating a new editor and deriving from VisBaseTriggerEditor. Look at VisAddForceTriggerEditor.cs or VisMessageTriggerEditor.cs for reference.

<u>Advanced Usage Walkthrough</u>

1. Create a Manager

a) Create an empty Game Object.

b) Add a Manager Script Component to your new game object from "Component=>Visualizer Studio=>Manager".

c) Configure the Manager component through the Inspector to match your usage requirements. Settings of a Manager are explained in the next section.

2. Add Data Groups

a) Add a Data Group Script Component to the same game object from "Component=>Visualizer Studio=>Data Group".

b) Configure the Data Group through the Inspector as needed to cover the desired spectrum range.

c) Repeat until all desired Data Groups are added.

3. Create Controllers

a) Add a Controller of any type to the same game object from

"Component=>Visualizer Studio=>Controllers".

b) Configure the Controller through the Inspector as needed to meet your desired functionality.

c) Repeat from step 'b' until all desired Controllers are added.

4. Add Modifiers and Triggers to Game Objects

a) Find any Game Objects that you desire to react to the music.

b) Add either a Trigger or Modifier to the Game Object from "Component=>Visualizer Studio=>Modifiers" or "Component=>Visualizer Studio=>Triggers"

c) Select the previously created Manager in the Inspector for this Modifier/Trigger.

d) Select the desired Controller in the Inspector that this Modifier/Trigger should connect to.

e) Configure the new Trigger/Modifier through the Inspector to make your Game Object react to the music as you desire.

Using Micrphone Input

1. Create VisMicrophone Component

a) Attach a VisMicrophone to the same GameObject as your AudioSource.

b) When the VisMicrophone component is active, it will automatically use that AudioSource as a microphone.

Support

Twitter

Visualizer Studio:	http://twitter.com/VisStudioUnity
Altered Reality:	http://twitter.com/AlteredR

Email

Visualizer Studio:	VisStudioUnity@yahoo.com
Altered Reality:	AlteredRealityEnt@yahoo.com

Website

Altered Reality:	www.alteredr.com

Credits

Lead Programmer and Architect

Andrew Thayer

Example Music by

Frankie Loscavio

Asset Store Images by

Xander Davis

Special Thanks

Jim Weinhart

Mitesh Shah

Leslie Thayer

Caiden Thayer

Software Disclaimer