# QTE SYSTEM

Documentation – v1.4

LAKEVIEW INTERACTIVE

# Contents

# Package Description and Features

Add Quick Time Events to your game – with no programming! QTE System provides a way to build seamless interactive narrative scenes.

Great for making interactive cinema, action scenes, and other games utilizing keyboard/controller prompts.

WebGL Demo

**Features:**

- **Easy-to-use:** change properties of your QTEs and **create your own QTE sequences without knowing how to program**
- **Six kinds of QTEs supported**: Single Press, Clickable, Multipress, Mashing, Alternating, and Sliding Time QTEs
- **Easy to customize**: Many options to tweak such as the color, images, and even what timer (Radial and Line) to use
- **Supports Keyboard, Controller, and Mobile input**
- **Works for 2D & 3D games:** Utilizes Unity's new GUI system so it works in both Screen Space and World Space as well
- **Complete and extensively documented C# source code included.**
- **Easy to follow** documentation
- **Several example scenes included**

## Update History

**1.4 (09.05.2022)**

- Added support to fail QTEs if player presses an incorrect button on Simple, Mashing, and Multipress QTEs
- Moved InputData to Resources folder for runtime loading. If updating your own project and you wish to use the **Fail if Incorrect** option, please move all valid options into a Resources folder or child of one.
- Documentation updates

**1.3 (04.02.2021)**

- Bug fixes for Line Timer prefab
- Addition of new Choice Menu QTE with additional Example

**1.2 (08.01.2021)**

- Bug fixes for QTE, MashingQTE, MultipressQTE and SimpleQTE classes
- Updated Documentation to use latest version of Standard Assets and FAQs
- Added Timeline Example with Timeline Resetter helper script

**1.1 (24.03.2020)**

- Added Alternating QTEs
- Adjusted Sliding QTEs to work in World Space
- Added background effects for all 2D levels
- Added "Get Ready" and "QTE Completed" events for Cutscene QTE Example

**1.0 (24.03.2020)**

- Initial version

### Credits

This asset uses the following third party open source libraries: iTween under the MIT license.

Controller Prompts were created by Nicolae "Xelu" Berbece and additional prompts for other controller types can be found at http://thoseawesomeguys.com/prompts.

The pattern for the background graphic used in the 2D example scenes is from www.toptal.com/designers/subtlepatterns/.

The Bungie Font is by David Jonathan Ross and is released under the SIL Open Font License.

## Overview of contents

Once imported in a Unity Project, you should see the following folders inside of the Lakeview Interactive folder:

- **QTE Examples:** This folder contains a series of scenes that can show off various aspects of the QTE System. Subfolders hold the legacy input system scenes as well as files used in the creation of the example scenes.
- **QTE System:** This folder contains several subfolders which make up the QTE System.
    - **InputData:** Holds the default input settings for the QTE System. Each QTE (aside from the ClickableQTE) requires at least one InputData in order to know what to press to complete it.
    - **Prefabs:** Holds all the prefabs used in the system. These are distributed into various folders for easier access.
    - **Scripts:** Holds all the scripts used by the base system. Each prefab contains one or more of these scripts.
    - **Sprites:** Holds all the sprites used by the QTE system.
        - **Inputs:** Holds sprites for controller, mouse, and keyboard input.
        - **QTEs:** Sprites created for specific QTEs.
- **ThirdParty:** This contains the iTween library which is used by the QTE System to animate timers, progress bars, and other UI elements used.

# QTE System Setup

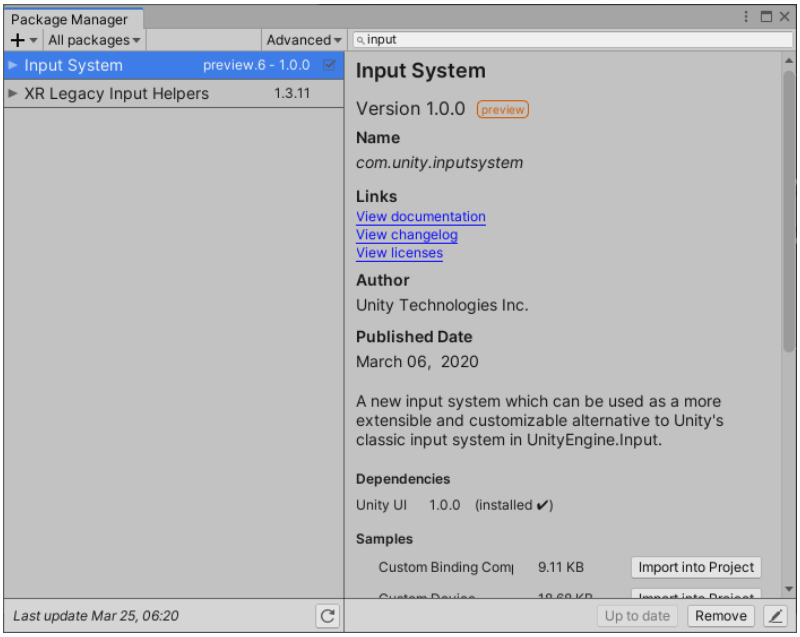
## Getting Started

The QTE System by default contains a number of project settings that may overwrite your project's current settings if you import it. With that in mind, it is a good idea to start with a blank project before importing the QTE System.

### Setup for new Input System

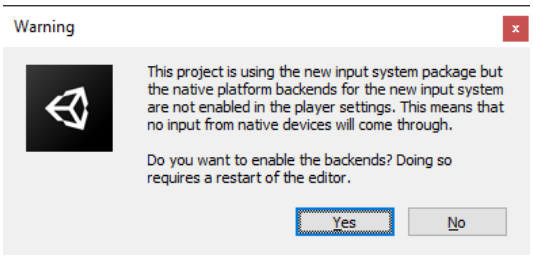
QTE System provides support out of the box utilizing Unity's Legacy Input system. QTE System also provides support for the new Input System that is currently in preview; but requires you to install it from the Package Manager. The steps to set it up follow:

1. Open up the Package Manager by going to **Window | Package Manager**.
2. From the Package Manager, ensure that you can see all of the options by changing the first dropbox to **All packages** and from the second dropdown on the right click on **Advanced | Show preview packages**.
3. Once that is done, select the **Input System** package and click on the **Install** but to add it to your project.




You may type "input" into the searchbar on the top right side to make it easy to see.

4. Once imported, Unity may give a warning. Click on the **Yes** button.



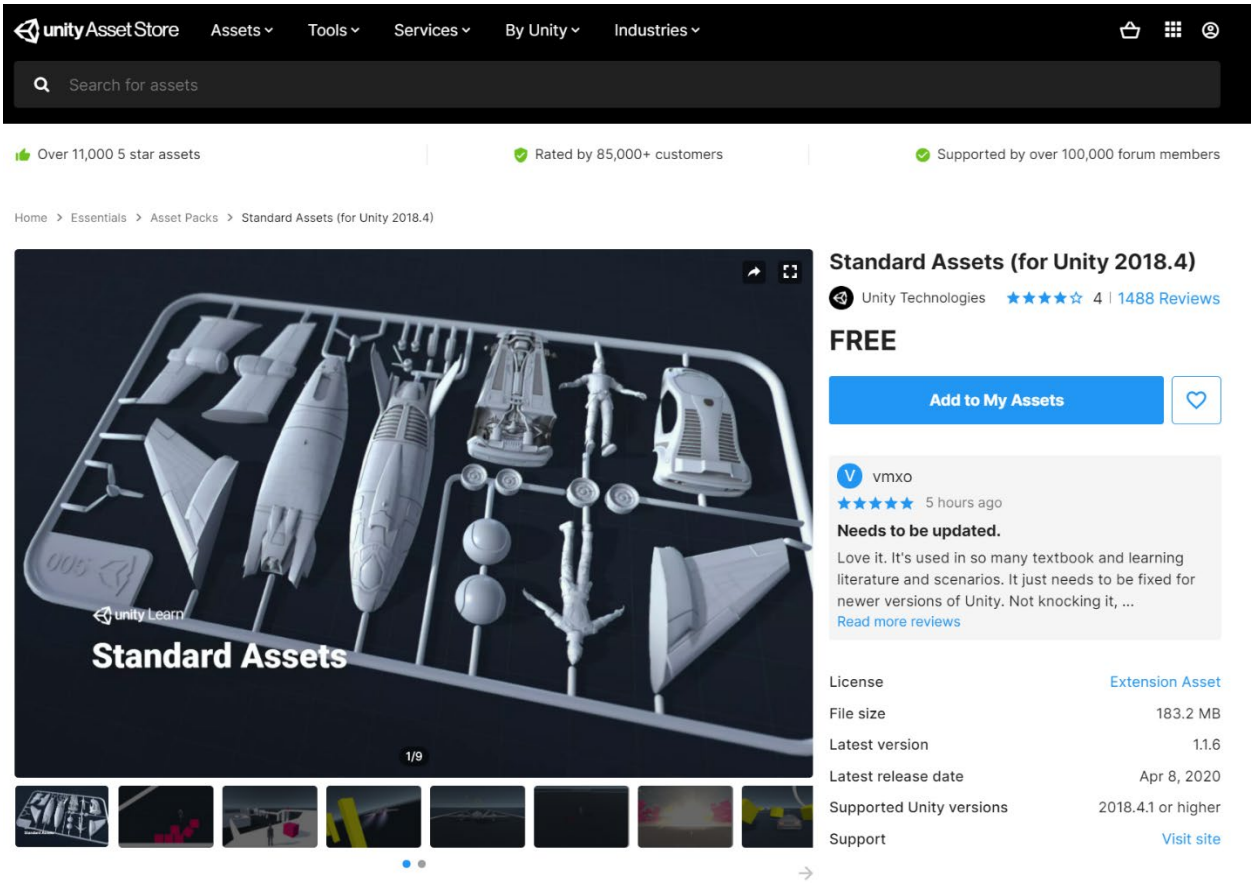

5. Afterwards, close your Unity Editor and start it up again.

Not doing so will cause null errors when you try to play the game.

At this point, the new Input System should be installed.

## Standard Assets Installation

To open one of the example scenes you will also need to download the freely available **Standard Assets (for Unity 2018.4)** package.




Specifically, the following folders are the only things that are needed:

- **StandardAssets\Characters\ThirdPersonCharacter** (Scripts & Prefabs subfolders are not needed)
- **StandardAssets\Vehicles\Car** (Scripts & Prefabs subfolders are not needed)

For convenience, you can also download a Unity package with the specific files you'll need at: http://johnpdoran.com/unity/AssetStore/QTESystem/QTE_StandardAssets.unitypackage

# Included QTEs

The QTE System comes with many built-in QTE types which you can utilize in your projects. A list of them follows:

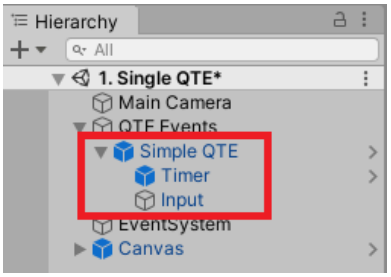
## Simple QTE

The simplest kind of QTE included with the project. Provides a single input. If hit within a certain period of time the player will succeed, otherwise they will fail.

The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Simple QTE contains the base object and two children:

**SimpleQTE** – Parent object which contains the SimpleQTE component which is discussed in detail below.

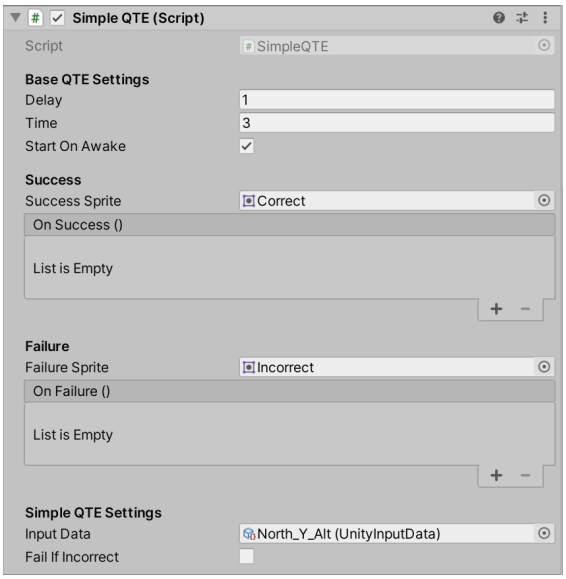



**Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

**Input** – An image which will display the input that should be pressed in order to succeed at the QTE.

### Properties

A list of the public properties that can be tweaked is as follows:




**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

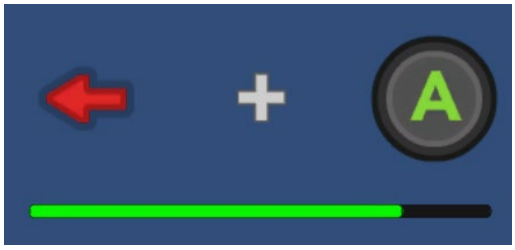
**OnFailure –** A list of callbacks which will be called upon the player failing to complete the QTE

**InputData –** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed

**Fail If Incorrect –** If enabled the player will fail the QTE upon pressing any of the other options from another InputData or key on the keyboard.

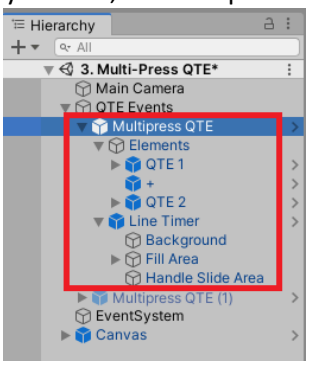
## Multipress QTE

This form of QTE makes the player have to press multiple buttons at once in order to trigger a successful hit. If all of the inputs are hit within a certain period of time the player will succeed, otherwise they will fail.



The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Multipress QTE contains the base object and two children:



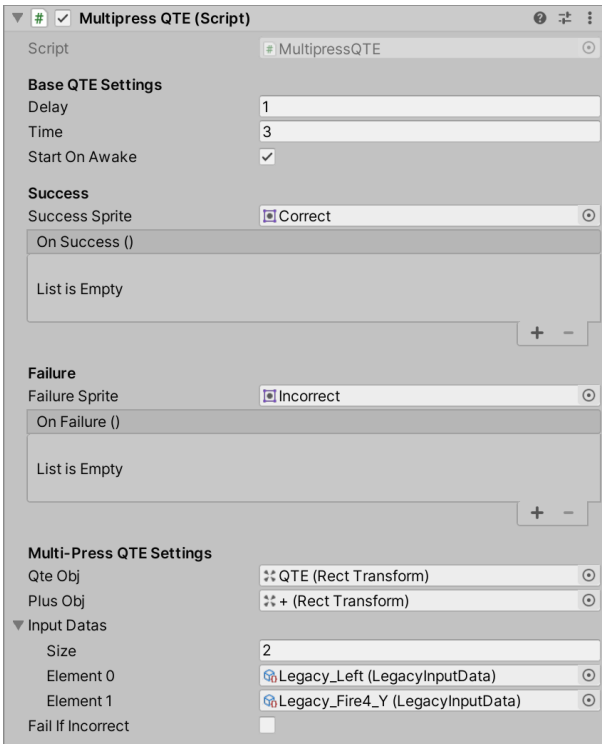

**MultipressQTE** – Parent object which contains the MultipressQTE component which is discussed in detail below.

**Elements** – This object will display all of the objects used to draw the Multipress QTE.

**Line Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

### Properties

A list of the public properties that can be tweaked is as follows:




**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

**Qte Obj -** An object reference to the QTE object in case it needs to spawn additional elements

**Plus Obj –** An object reference to the + object in case it needs to spawn additional elements

**InputDatas -** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed. While the examples only show two elements, it is possible to add more and the system will automatically expand to hold all of them.

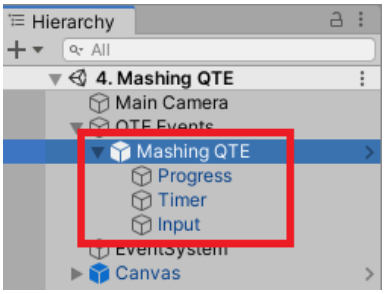
**Fail If Incorrect –** If enabled the player will fail the QTE if they press a key that is not from any of the given InputDatas. Basically, they can't press anything but the correct combination to continue.

## Mashing QTE

This form of QTE requires the player to press the same input multiple times in order to fill a progress bar. Upon reaching the end of the progress fill, the player will trigger a successful hit.

The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Mashing QTE object contains the base object and three children:




**MultipressQTE** – Parent object which contains the MultipressQTE component which is discussed in detail below.

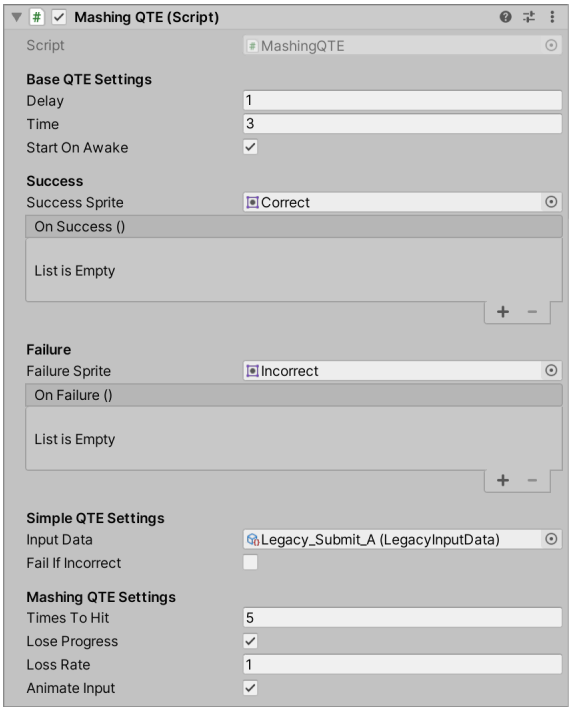
**Progress** – This image will increase and decrease in size based on how close the player is to finishing the QTE.

**Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

**Input** – An image which will display the input that should be pressed in order to succeed at the QTE.

### Properties

A list of the public properties that can be tweaked is as follows:




**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

**InputData -** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed

**Fail If Incorrect –** If enabled the player will fail the QTE upon pressing any of the other options from another InputData or key on the keyboard.

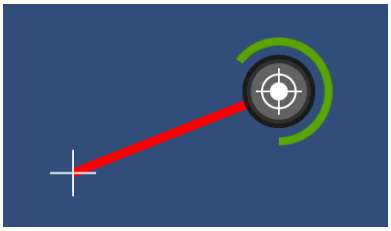
**Times To Hit** – Assuming no lost progress, how many times would the player need to press the input in order to trigger a success.

**Lose Progress** – Should the player lose progress over time?

**Loss Rate -** The rate at which the player loses progress, the higher the number the harder the QTE

**Animate Input** – If enabled, will have the input button animate to simulate button mashing.

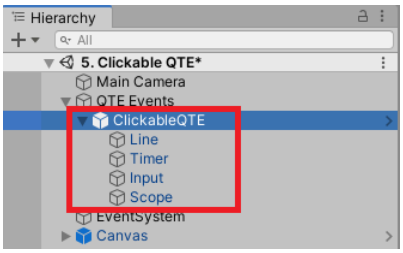
## Clickable QTE



Unlike the other QTEs this requires the player to use their mouse and/or move the scope on the screen to the target and activate it. Upon activation, the QTE will trigger a successful hit.

The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Clickable QTE object contains the base object and four children:




**ClickableQTE** – Parent object which contains the ClickableQTE component which is discussed in detail below.

**Line** – A line that will draw from the scope's position to the target. Note: This should only be used in Screen Space.

**Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

**Input** – An image which will display the input that should be pressed in order to succeed at the QTE.

**Scope** – A visual representation of the mouse's position. When using controller this will move through the use of the left analog stick

## Properties

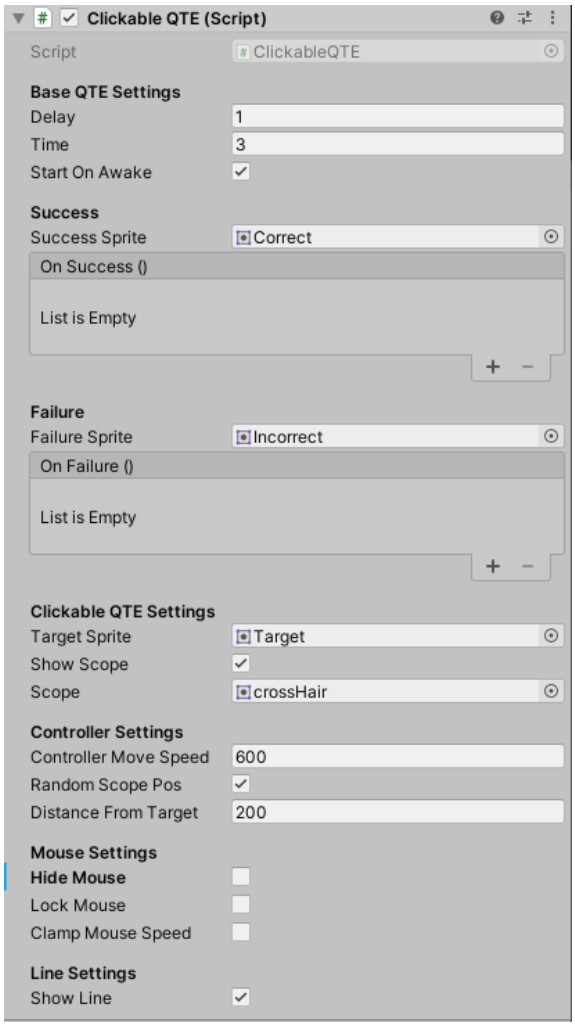
A list of the public properties that can be tweaked is as follows:

**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time




**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

**Target Sprite –** What sprite should be used on the target

**Show Scope –** Should the scope object be displayed?

**Scope -** The cursor that will be shown on the screen while the QTE is active

**Controller Move Speed -** How many pixels per second the controller should be able to travel

**Random Scope Pos -** Should the scope be moved to a random position before starting the QTE?

**Distance From Target -** If randomControllerPos is enabled, how far the scope should be from the target

**Hide Mouse -** Should the mouse cursor be displayed?

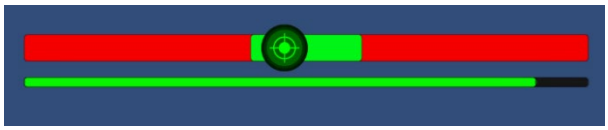
**Lock Mouse -** Should the mouse be locked in place during the QTE. Locking the mouse position for the duration of the QTE will have the player use the scope's offset for movement.

**Clamp Mouse Speed -** Should the scope's speed by limited when using mouse input?

**Show Line -** Should the line be displayed or not? Note: Does not work correctly on Mobile or in World Space

## Sliding QTE

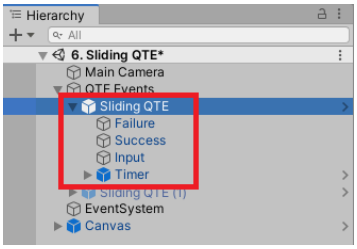
Rather than just pressing the input at any time, the Sliding QTE requires the user to also time when the press the button correctly. Upon activation, the QTE will trigger a successful hit.



The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Sliding QTE object contains the base object and four children:

**SlidingQTE** – Parent object which contains the ClickableQTE component which is discussed in detail below.

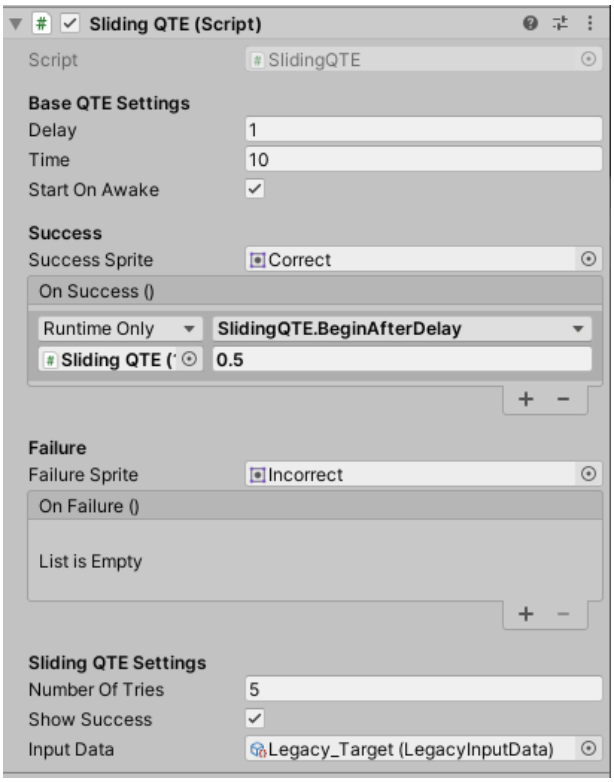



**Failure** – The entire portion of the failure state. Can be moved and scaled using Unity's UI tools to make the QTE easier or harder.

**Success** – The portion of the QTE that is consider to be a success. Can be moved and scaled using Unity's UI tools to make the QTE easier or harder.

**Input** – An image which will display the input that should be pressed in order to succeed at the QTE.

**Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.




### Properties

A list of the public properties that can be tweaked is as follows:

**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

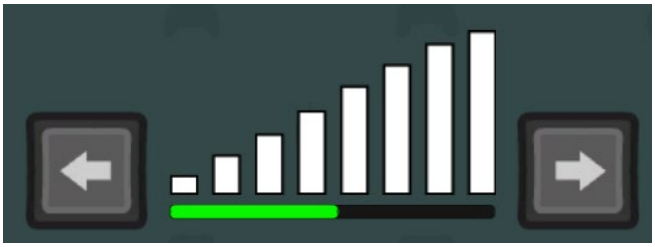
**Number of Tries –** How many times will the slider move from one side to the other.

**Show Success –** If the success sprite should be shown upon completion or not.

**InputData -** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed

## Alternating QTE

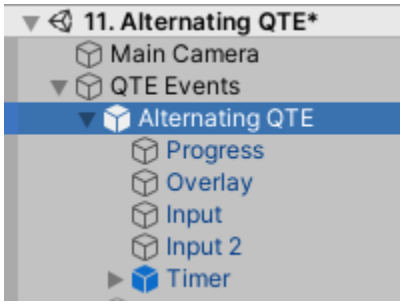
This form of QTE requires the player to press the same input multiple times in order to fill a progress bar. However, unlike the Mashing QTE, this QTE requires the user to press alternating buttons which can be used to simulate running or doing some other kind of strenuous action. Upon reaching the end of the progress fill, the player will trigger a successful hit.



The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Mashing QTE object contains the base object and three children:

**AlternatingQTE** – Parent object which contains the AlternatingQTE component which is discussed in detail below.




**Progress** – This image will increase and decrease in size based on how close the player is to finishing the QTE.

**Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

**Input** – An image which will display the first input that should be pressed in order to succeed at the QTE.

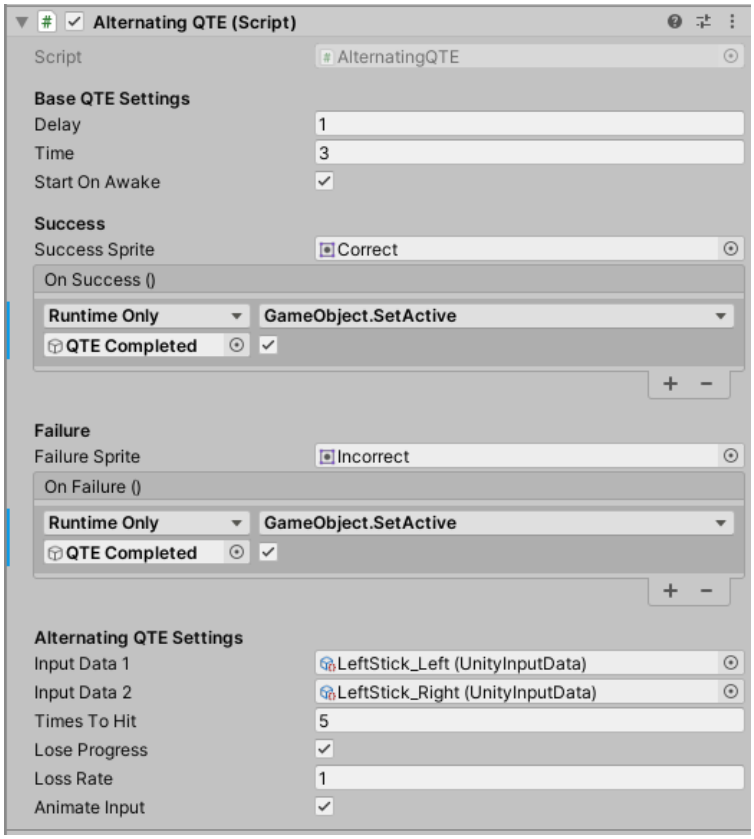
**Input2** – An image which will display the other input that should be pressed in order to succeed at the QTE.

## Properties

A list of the public properties that can be tweaked is as follows:

**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last




**Start on Awake –** Should the QTE start immediately?

**Success Sprite -** The sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

**Failure Sprite –** The sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

**InputData1 -** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed for the Input object

**InputData2 -** The information used by the QTE in order to determine what input sprite to show and what button/key to press in order to succeed for the Input 2 object

**Times To Hit** – Assuming no lost progress, how many times would the player need to press the input in order to trigger a success.

**Lose Progress** – Should the player lose progress over time?

**Loss Rate -** The rate at which the player loses progress, the higher the number the harder the QTE

**Animate Input** – If enabled, will have the input button animate to simulate button mashing.

## Choice Menu QTE

This QTE will allow users to select one from a series of possible choices within a certain time limit. If a choice was picked the QTE counts as a success, if none was select it counts as a failure. Uses Unity's UI system and the input system with that to automatically allow you to select via mouse/keyboard or controller.


The QTE is a UI object so it is required to be a child of a Canvas object.

By default,the Choice Menu QTE contains the base object and two children:

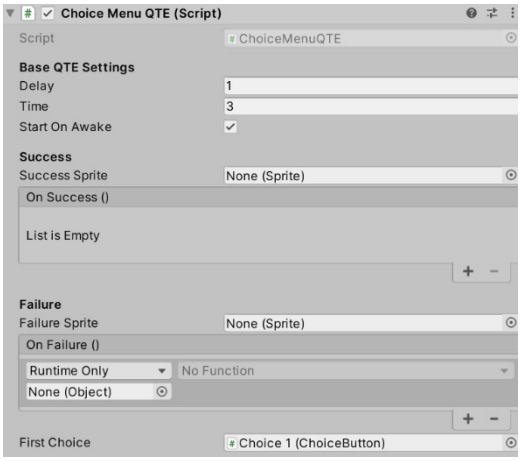
**Choice Menu QTE** – Parent object which contains the ChoiceMenuQTE component which is discussed in detail below.


**Choice (1/2/etc)** – An option that can be picked within the QTE itself. It is possible to create more by duplicating or creating other buttons using the Choice Button class.

**Line Timer** – A visual representation of the time elapsing over the course of the QTE. This object may be removed if not desired.

### Properties

A list of the public properties that can be tweaked is as follows:




**Delay –** How long to wait until the QTE starts

**Time –** After starting, how long should the QTE last

**Start on Awake –** Should the QTE start immediately?

**Success Sprite –** (Not used) If there is a child named Input, the sprite that the input will change to upon the player pressing the correct input in time

**OnSuccess –** A list of callbacks which will be called upon the player successfully completing the QTE

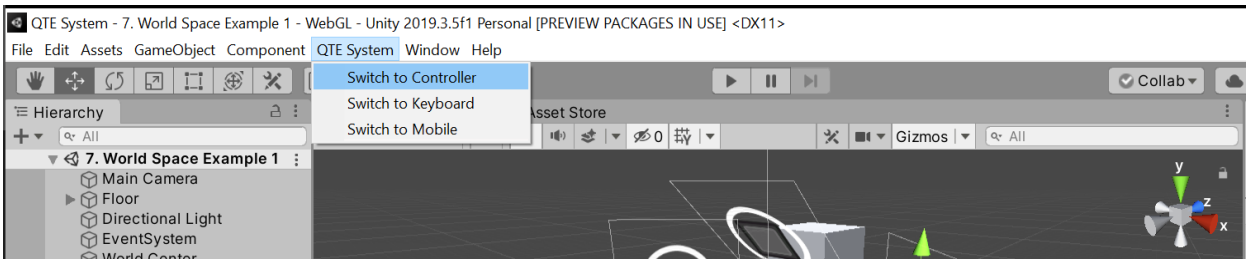
**Failure Sprite –** (Not used) If there is a child named Input, the sprite that the input will change to upon the player not pressing the correct input in time

**OnFailure-** A list of callbacks which will be called upon the player failing to complete the QTE

**First Choice –** The button (if any) that will be selected whenever the QTE is started

# Input Mode

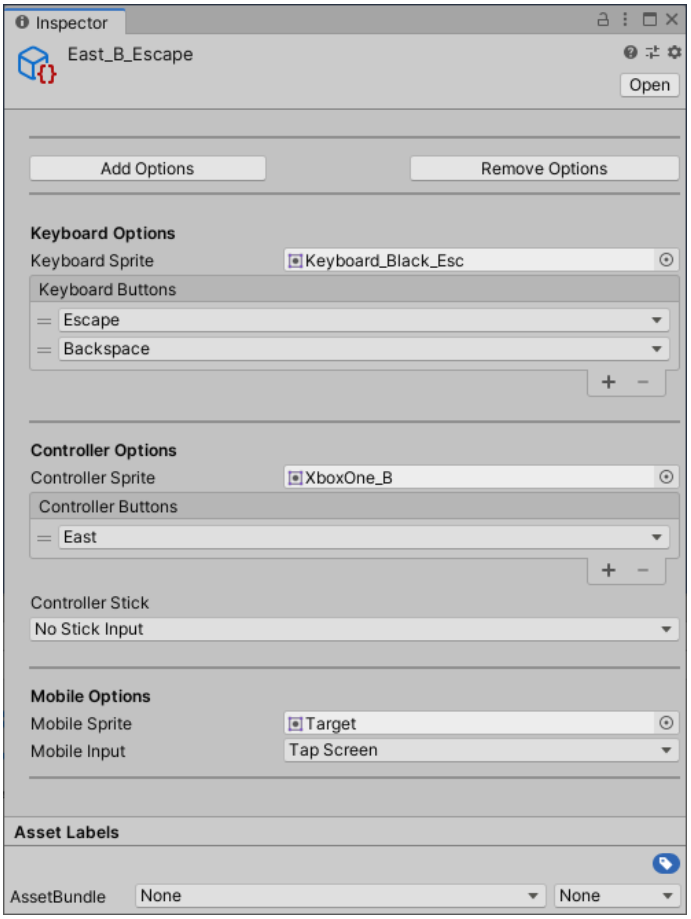
QTE System is built in such a manner that it can support keyboard, controller, and mobile input. It is possible to set the current mode through code through the QTE.inputMode property or by selecting from the top menu the QTE System menu and selecting one of the options.




It is important to note that the mobile input only works on mobile devices in this current version of QTE System.

# Input Data

In order to support controller, keyboard, and mobile input we need to know what option we should use per platform. Since this will be reused often this has been turned into its own type called an InputData. The InputData objects are located inside of the **QTE System\InputData** folder.




There are a number of different properties which can be tweaked.

**Keyboard Sprite** – The sprite that will be displayed on the QTE input when this input mode is chosen.

**Keyboard Buttons** – List of all of the valid inputs that are used when the keyboard input mode is chosen.

**Controller Sprite** – The sprite that will be displayed on the QTE input when this input mode is chosen.

**Controller Buttons** – List of all of the valid inputs that are used when the controller input mode is chosen.

**Mobile Sprite** – The sprite that will be displayed on the QTE input when this input mode is chosen.

**Mobile Input** – The kind of mobile input (if any) that should be accepted for this method if the game is deployed on iOS or Android.

While a number of inputs have been created and provided by default it is also possible to create your own by going to the **Project** window and right click and select **Create | QTE System | Unity Input Data**.

The easiest way to add valid input is to select the **Add Options** button and then start pressing the keys and moving the controller with what you would like to accept. The system will automatically add those items to your selection. Click on the **Cancel** button when finished. The **Remove Options** button works in a similar fashion, removing selections if they are currently there.

# Creating a QTE Sequence

It is quite easy to create a custom QTE sequence depending on what you wish to have happen in your game. We will first discuss how to create a QTE sequence utilizing a single Canvas object in Screen Space and then build upon that knowledge to create QTEs in World Space as well.

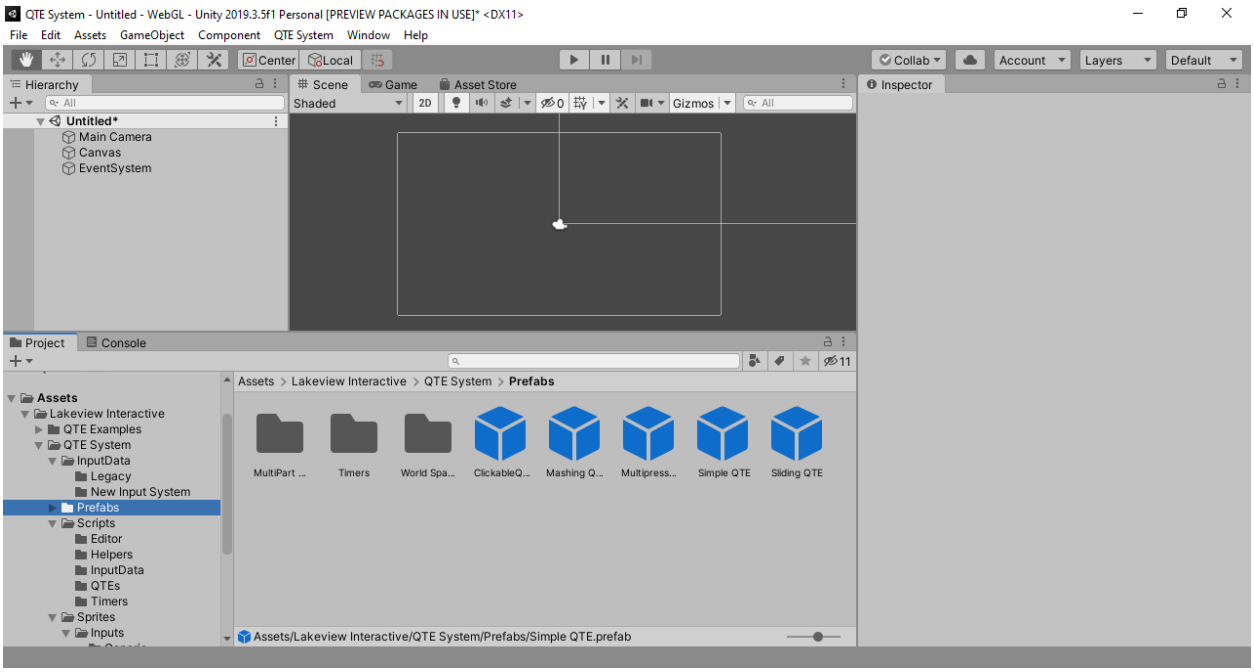
## Screen Space

To get started with Screen Space the first step will be to create a Canvas object if you have not done so already.

1. From the menu select **GameObject | UI | Canvas**.

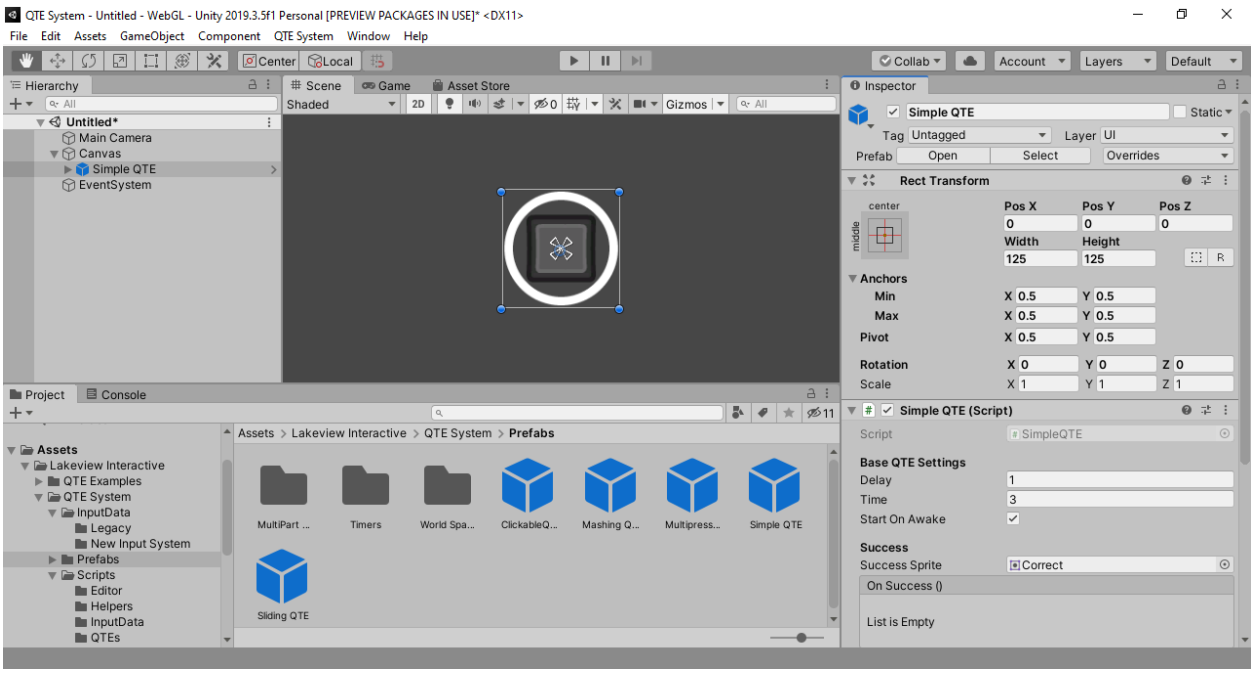
This will create both a Canvas and an Event Listener game object. If you are using a ClickingQTE you must have the EventSystem in your scene; but it is not required for any of the others.

2. Then from the Project view go to the **Lakeview Interactive\QTE System\Prefabs** folder. From there you will see a number of different selections.




3. From there, drag and drop any of the QTE prefabs onto the Canvas object.

If the object is not a child of the Canvas it will not appear on the screen.




If all went well, you should be able to double-click on the QTE object within the Hierarchy and it will zoom into the position of the object.

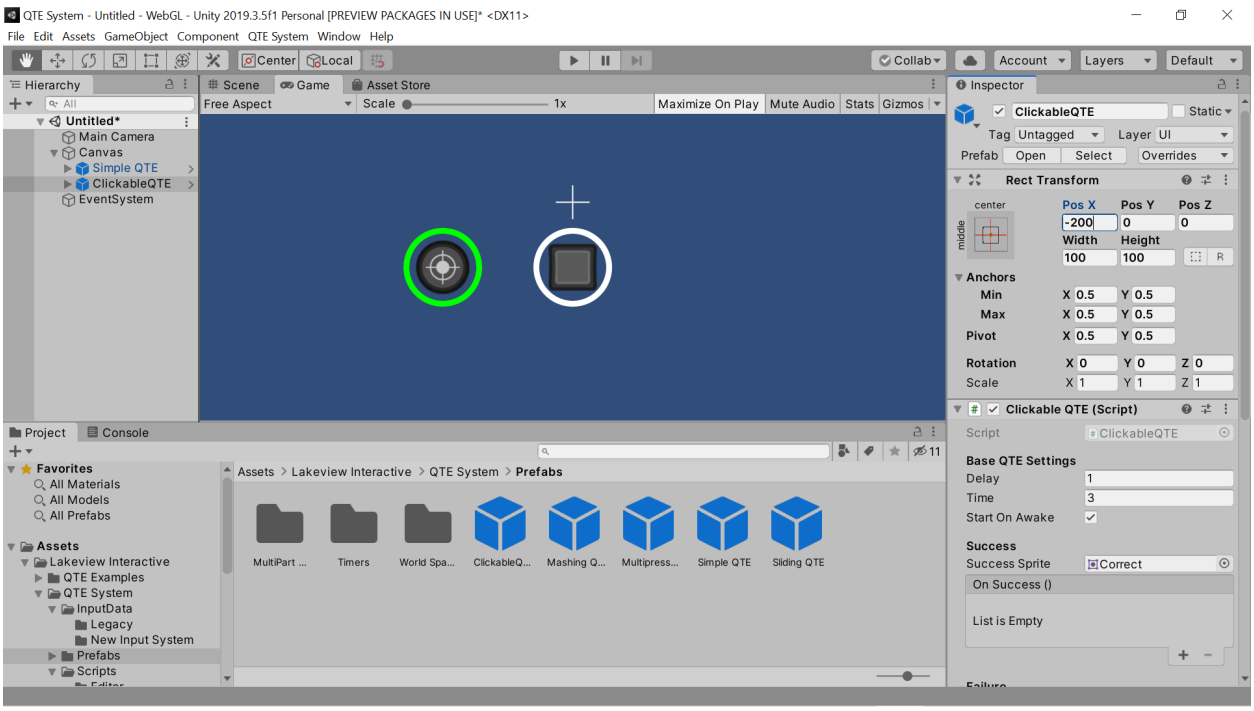
4. Once in the scene you may move the object using the transform tools or modifying the **Rect Transform** component from the **Inspector** tab.

For those who would like to see more details on modifying UI objects in Unity as well as what the properties on the Rect Transform mean check out: https://www.febucci.com/2018/10/unity-ui-tutorial/

5. Once your QTE is positioned where you'd like it you can customize the properties as you would like. In particular the **Input Data** property which dictates what keys/controller inputs are considered to be successful. For more information on the Input Data object, check out the Input Data section of this tutorial.
6. Next, let's add another QTE to our scene as well.

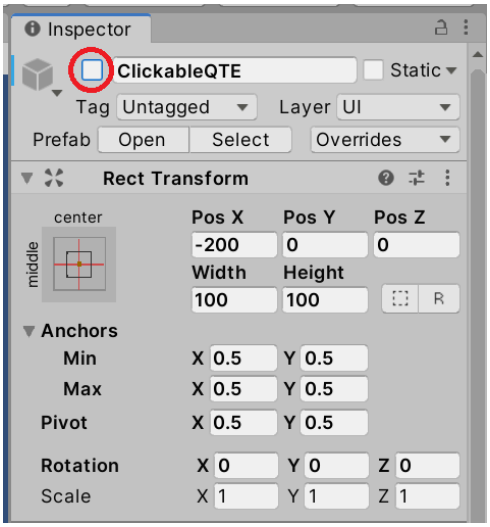
This can be the same as the previously created QTE or any of the others.

7. Once added, move the QTE so that it is not overlapping with the previously created one if needed.



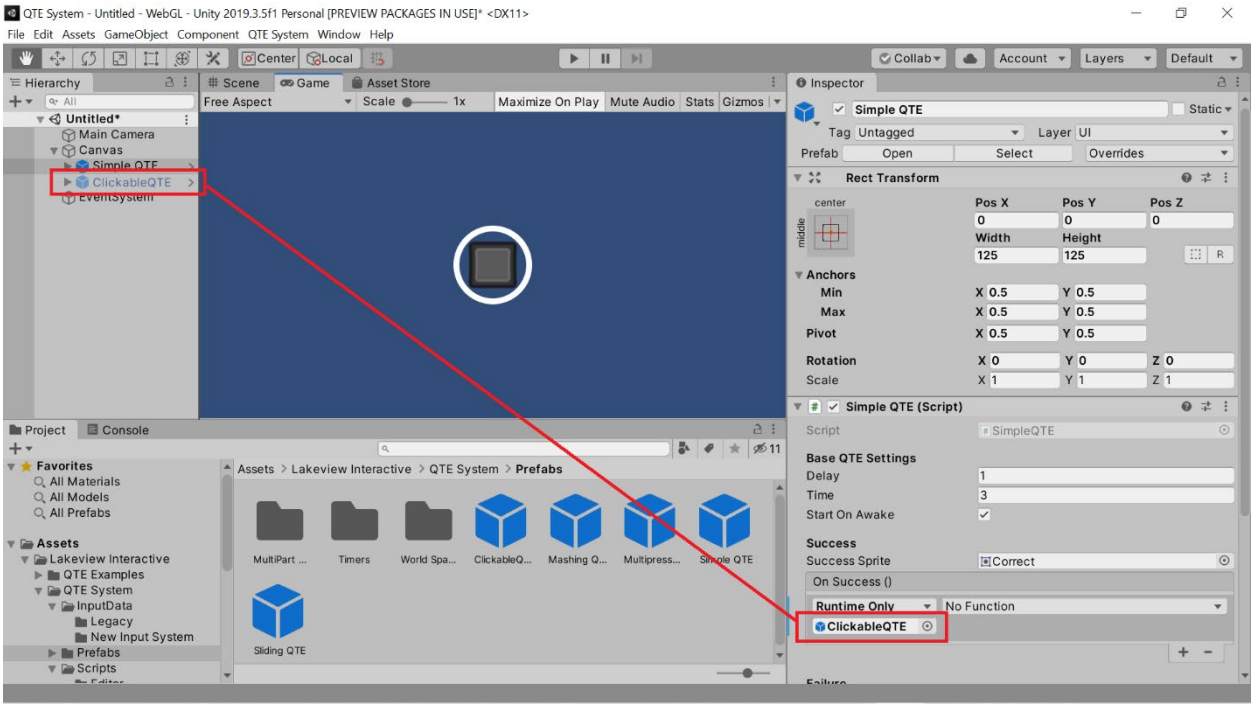

If you play the game right now, both QTEs will start at the beginning of the game.

8. If you wish to have one start after the other has finished, then turn off one of the two QTEs by going to the **Inspector** window and clicking on the check next to its name.




This will remove the object from the scene. Now we can make it so that it will turn on after the first one has completed.

9. Select the QTE that is still enabled. Once there, scroll down to the **On Success ()** property and click on the **+**. From there, in the window drag and drop the disabled game object from the Hierarchy window into the slot.



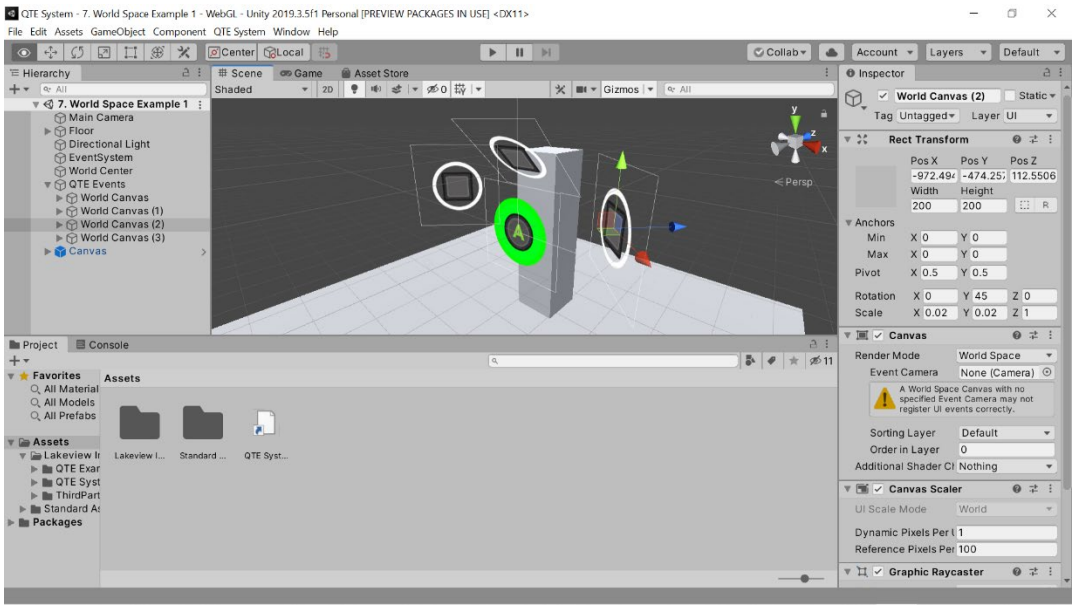

10. Then click on the dropdown that says **No Function** and then select **Game Object | Set Active**. Once there click on the check box to turn it on. Once finished it should look like the following image:


11. Save your scene and try the game!


Upon finishing the first QTE, the second one starts! Doing the same steps, it is possible to make your QTE event as long as you'd like and customize it to have different properties or even different things to happen upon failing the event.

## World Space

Creating a QTE in World Space is much like creating one in Screen Space aside from the fact that elements need to have their own Canvas and are contained in the **QTE System\Prefabs\World Space** folder.

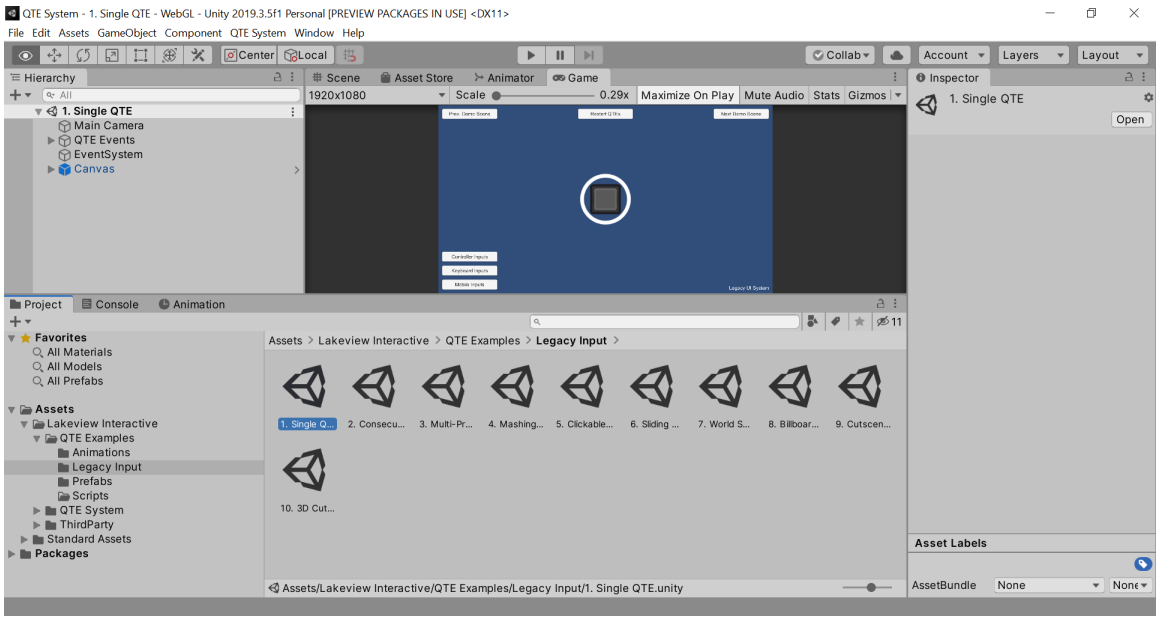


# Using Unity's Legacy Input System

Unity's legacy input system is set up to utilize the Input.GetButtonDown and Input.GetAxis functions in order to detect input. For those that would still like to use this form of input, all the files utilizing it are included with the QTE system.

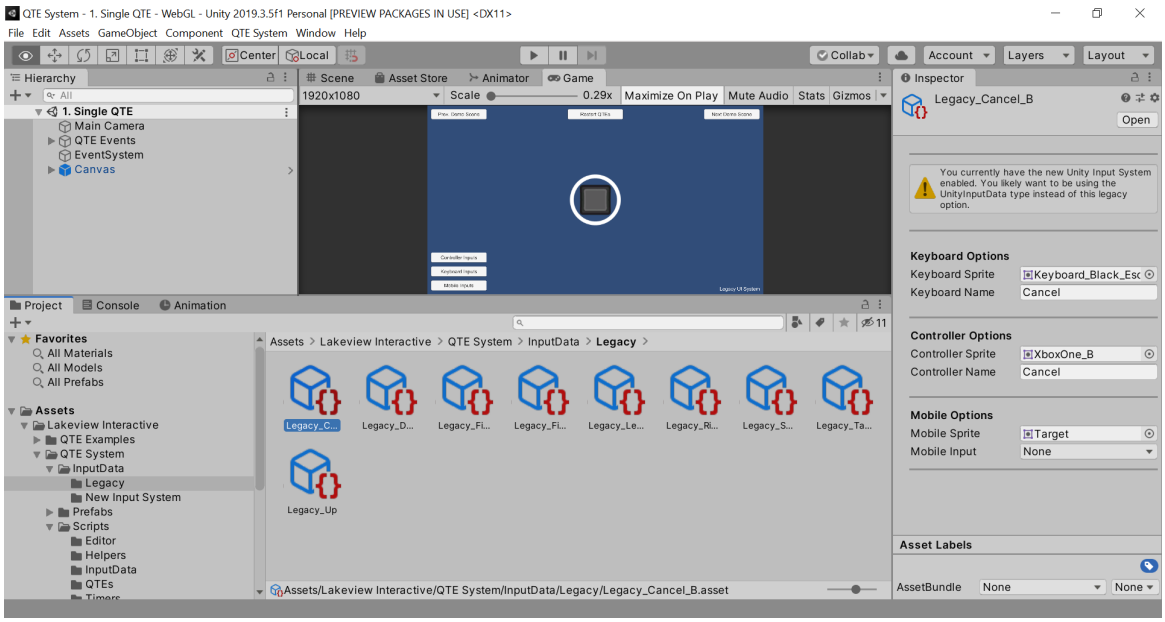
## Scene Files

You can access the demo scenes for this input system by going into the **QTE Examples\Legacy Input** folder:



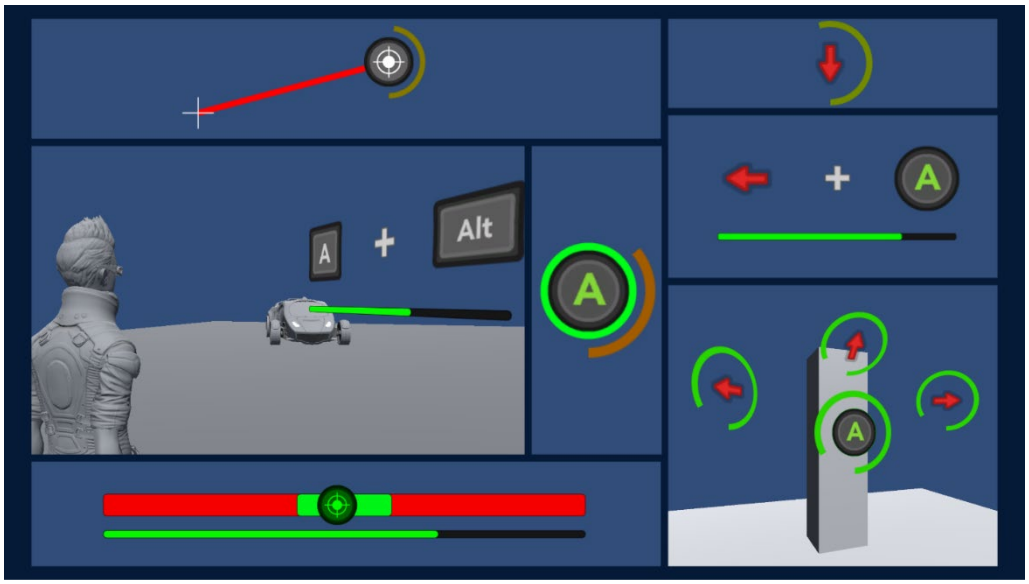
## Legacy InputData Objects

The InputData objects are located inside of the **QTE System\InputData\Legacy** folder




The included legacy InputData is set up to work correctly on Windows with an Xbox 360 Controller, but it can easily be tweaked to work for other controllers and platforms.

More information on setting it up for Mac OS X and Linux can be found at:
https://wiki.unity3d.com/index.php/Xbox360Controller



# Frequently Asked Questions

## General

**I am stuck and need help, what can I do?**

If you can't find a solution for your problem in this doc, contact me! I will gladly help to solve your issue.

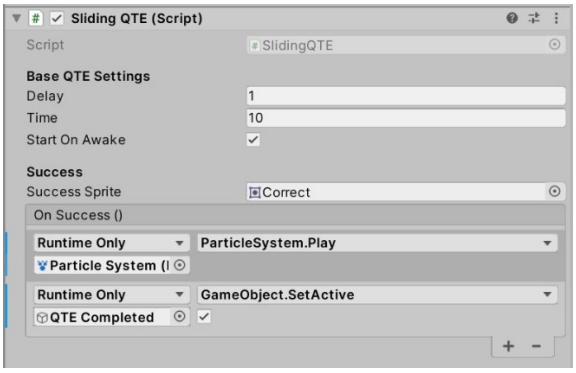
**Does QTE System run on mobile platforms?**

Yes, it does. When deployed on a mobile device the game will utilize the Mobile Settings section of the Input Data provided. When creating QTEs in Screen Space, it is a good idea to utilize the Canvas Scaler component that scales with the screen size and anchors (you can find more information about Unity's UI system here).

**How can I have gameplay events happen after the QTE is completed?**

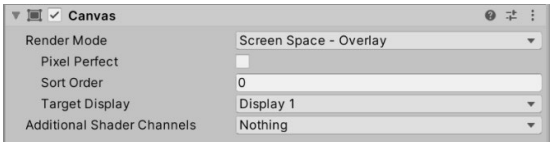
To begin, you will need to write a public function on some component that has the action you want to have happen and attach that to some game object within your QTE's scene.

Once completed, select the last QTE object in your QTE sequence and from the QTE script go to the **On Success** property. From there click on the **+** icon. There is an example of this in the **QTE Examples** folder under **9. Cutscene Example** as upon completion of the QTE a particle system will be played, and text will show that the QTE has been completed.



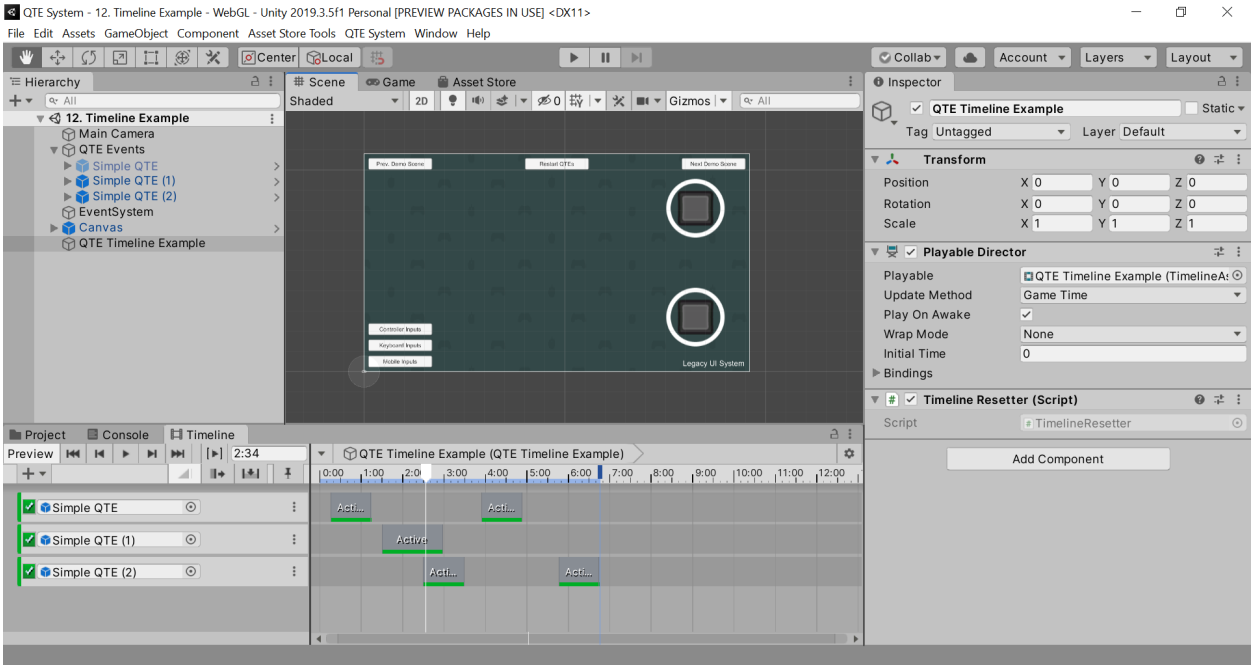

**Can you use the QTE System with Cinemachine?**

Yes, you can. When you create QTEs ensure that you have a Canvas object as their parent. As long as the **Render Mode** property on the **Canvas** component is set to **Screen Space - Overlay** the QTE Systemshould work perfectly with Cinemachine and be placed on top of the screen in front of whatever is on the camera.



**Can you use the QTE System with Timeline?**

Yes, you can. There is an example of this within the QTE Examples folder under **12. Timeline Example**. It is possible to use the Timeline editor to start QTE events by utilizing activation tracks. Basically, each of the QTE Events would be turned off at the start of the game and upon starting the Timeline when the QTE is set to be active it will appear on the screen.




In terms of each QTE, you will want to note how long you are planning the QTE to be on screen and adjust the **Delay** and **Time** properties to ensure the time adds up to the entirety that the QTE is on screen.

**What is your policy on refunds?**

As a general policy, we do not accept refunds on assets that have already been downloaded because of the following reasons:

- There is no way for the assets to be digitally "returned".
- We feel that, between the playable demo, the description, the official documentation and the detailed screenshots, there is enough information available to customers prior to purchase. Of course, we are also always happy to answer any pre-purchase questions.
- We do our best to fix any bugs or issues reported in subsequent updates.

## Building for iOS and Android

**How do I make a build for iOS?**

You need to follow the official instructions in the Unity manual here.

**How do I make a build for Android?**

You need to follow the official instructions in the Unity manual here.

## Licensing

**What are the licensing terms of the kit?**

The QTE System uses the default Unity Asset Store End User License Agreement. A copy of the Asset Store EULA is available at: https://unity3d.com/es/legal/as_terms

Thanks so much for purchasing this system! I hope you will let me know what you're doing with it!