Table of Contents

Description and Features

Thank you for purchasing Quick Time Event system for Unity, this is a system for having QuickTime events in your game. This document describes in detail all options available in the system, as well as how to set it up.
Also an example scene is provided to demonstrate most of the capabilities in action.

Included with the system is a demo scene that shows all the available options the system has already set up. This document goes into detail on how to set them up yourself and explains all the options.

Configurable options

- Can define QTEs asking for standard keyboard input, or use Buttons or Axes defined in Unity's Input manager
- Have the script choose a random input to ask for from all available, or from a list of random inputs to choose from, or define a static one the QTE will always use
- QTEs can be Repeatable, or single shot

- Choice between Single, Dual, Tri or Quad buttons for the player to choose from
- Mashable QTEs, when the player must mash a single button enough times. With visualization of the mashing struggle.
- Separate Failure detection: Running out of time, and optionally for pressing the wrong button.
- Visualize the Time the player has to complete the QTE via radial wiping circle
- Uses Unity 4.6+ uGUI system, Buttons can be used in any style Canvas (Screen space, Camera Space, World Space)
- Script provided so World Space buttons can orientate to look at an object.
- Script provided so World Space buttons move along with an object.
- Can make the QTE "invisible" by hiding the on screen UI buttons from displaying, allowing to you show your own objects/UI instead
- Have buttons shake back and forth
- Response scripts that can be stacked to cause multiple things to happen in response to the QTE result
- Ability to extend the response options to the QTEs by writing your own custom response scripts.

Upgrade Notes:

If you are upgrading from a previous version of this system.

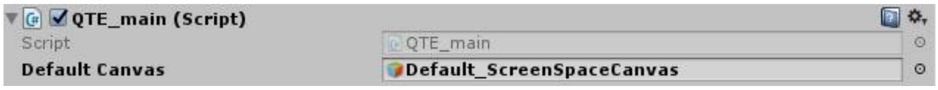
1. You first should delete the "QTE_System" and "Resources/QTE_System" folders before importing this new version
2. Complete the "Main Script & Default Canvas" Section.
3. Unfortunately, **all Triggering scripts saved into scenes or prefabs will have their values reset to defaults.** This is because of the code base received an optimized overhaul in which many values were renamed. So you will have to set the values in them up again.
4. The "Billboard" Buttons (which were meshes being spawned in the world) functionality has been removed, because instead Unity's Canvas' can be set to world space to accomplish the same effect. Look at the World Space Canvas Section to set it up.
   Therefore, the meshes/prefabs and the following two scripts have been removed, and therefore will have to be removed from any scene/gameobject that was using them. "QTE_Billboard_Lookat.cs" and "QTE_Billboards.cs"

Documentation

The System is broken up into 3 sections.

- **The Main script** - Contains all the logic, (getting user input, determining the result, Timers, showing the textures etc.) does not actually do anything until triggered.
- **Triggering Scripts** - the Script that actually causes a Quick Time Events to happen, also configures exactly what the QTE is (single, dual, mashable, what button to use etc.)
- **Response Scripts** - Scripts that respond to the result of the QTE, and perform an action in the scene.

The Main script & Default Canvas



☑ QTE_main (Script)

Script: QTE_main

Default Canvas: Default_ScreenSpaceCanvas

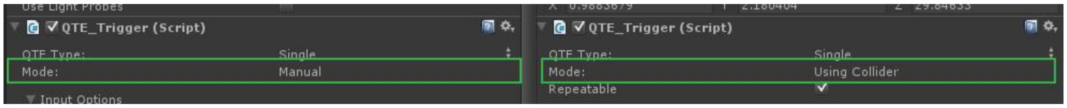
1. To get started, the first step is to drag the " QTE_Main" prefab from "\QTE_System\Prefabs\" into your scene.
2. You will now need to create a Canvas in your scene to act as the default canvas that the system uses. This Canvas can be set up anyway you'd like.
3. After creating the canvas, drag the "QTE_UI" prefab from"\QTE_System\Prefabs\" into your created Canvas to setup the necessary gameobjets the system uses.
   From there you can adjust the default position and scale of the buttons. They will appear this way unless you choose to override their positions later.
4. Finally, select the "QTE_Main" Prefab in your scene, and in the QTE_Main script, assign the Canvas you created to "Default Canvas".

Triggering Scripts

The Triggering script "QTE_Trigger.cs" actually causes Quick Time Events to happen, it can be found in "\QTE_System\Scripts\Triggers".

You attach this script to any gameobject in the world, and it fires Quick Time Events when triggered.

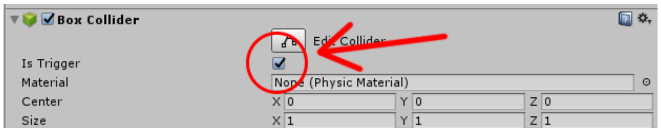
There are two ways of triggering QTEs to happen "Collider" and "Manual".
At the top of the script "Mode" will tell you which style it is using.



QTE_Trigger (Script)
QTE Type: Single
Mode: Manual
Input Options

QTE_Trigger (Script)
QTE Type: Single
Mode: Using Collider
Repeatable ✓

- **Collider Mode**
  When the triggering script is attached to a GameObject that has a Collider **AND** "Is Trigger" set to true, the QTE will fire when the player (The GameObject tagged as "Player") walks into the collider.

  Box Collider
  Is Trigger ☑
  Material None (Physic Material)
  Center X 0 Y 0 Z 0
  Size X 1 Y 1 Z 1

  If your Player does not use the Character Controller, and instead uses a collider for Collison, be sure to attach a rigid body and freeze its rotations. Or else your player may not be able to trigger the Quick Time Event when they walk through the collider.

- **Manual Mode** is set up to be called by any other method that you wish to use listed below
  But with all the different methods listed below the main idea is you just need to access the triggering script, and call TriggerQTE()

  A. By calling the function TriggerQTE() in your own script.
  Note that if you are doing this in the update() function, you only want to do this once in a frame. If you are constantly setting/calling the QTE to happen every frame you will find that the QTE will fire, but you'll be unable to complete it as the button input won't work (as it is constantly firing a new Quick Time).
  B. Via Animation Events in the Animator editor, or via Animation events on imported clips (call the function TriggerQTE()).
  You need to attach a triggering script the same GameObject that has the animation event to trigger it.
  C. Using Animator Curves by using another included script to trigger them.
  D. a Cutscene or animation editor (such as Unity's animation window, uSeqencer, Aperture Cutscene Editor, or CutScene etc)
  For this option you can attach the Triggering script to any GameObject in the scene (I recommend empty GameObjects to avoid any conflicts and easy recognition), then use the Editor to animate the boolean TriggerEnabled value to true, then immediacy false again.

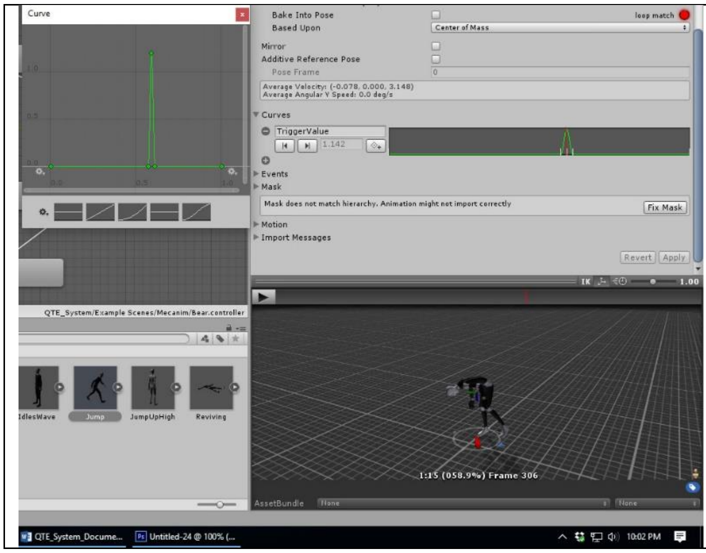
Interrupting QTE's

It is possible to stop a any QTE's in progress by accessing the main script, and calling InteruptQTE() This will cancel out any QTE, remove any UI from the screen and no responses will happen.

Triggering by Animation Curves

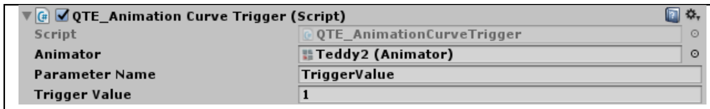
You can optionally trigger QTEs via Animator Curves.
An example of how you would use this is setting up a curve for a punching animation, so that a QTE fires in sync at a specific time of the animation such as a character winding up to punch.
To learn how to set up an animation Curve, watch this section of this tutorial.

1. Create a curve that Spikes above the value you want to cause the QTE to fire, at the time you want it to fire. Make sure it "spikes", because if the curve remains above the value for more than a brief time, the QTE will fire multiple times.



Here you can see I set up a curve to Spike above the value of 1 when the Character lands

2. Attach "QTE_AnimationCurveTrigger.cs" to the same GameObject as the Trigger script.

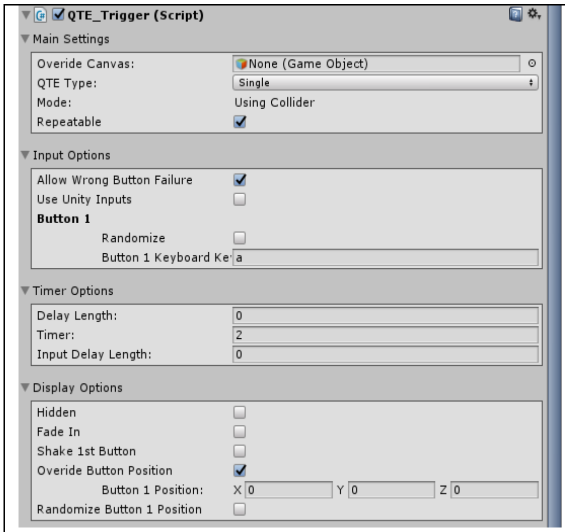

QTE_Animation Curve Trigger (Script)

Script	QTE_AnimationCurveTrigger
Animator	Teddy2 (Animator)
Parameter Name	TriggerValue
Trigger Value	1

Animator Object	The GameObject with attached Animator component
Parameter Name	The name of the Float Parameter that is being changed by a Curve.
Trigger Value	The value the float has to be greater than or equal to, to cause the QTE to fire.

When the game runs, and the specific animation plays that has a curve that animates the specified parameter and it reaches or goes over the "Trigger Value", the QTE will fire.

Triggering Script Options:



☑ QTE_Trigger (Script)

Main Settings

Overide Canvas: None (Game Object)
QTE Type: Single
Mode: Using Collider
Repeatable ☑

Input Options

Allow Wrong Button Failure ☑
Use Unity Inputs □
Button 1
Randomize □
Button 1 Keyboard Ke a

Timer Options

Delay Length: 0
Timer: 2
Input Delay Length: 0

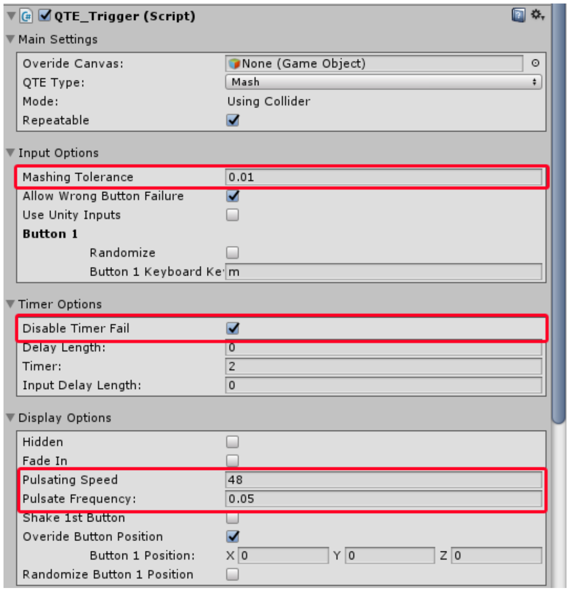
Display Options

Hidden □
Fade In □
Shake 1st Button □
Overide Button Position ☑
Button 1 Position: X 0 Y 0 Z 0
Randomize Button 1 Position □

Main Settings	
Override Canvas	If left empty, the Default Canvas assigned in the main script will be used. If another canvas is specified here, it will be used instead.
QTE Type	Changes the type of the QTE, (Single, Dual, Tri, Quad, or Mashable).
Mode:	"Collider" if attached to a gameobject with a Collider, "Manual" otherwise
Repeatable (Collider only)	If true, the QTE will always fire every time the player enters the collider, if false, only fires the first time.
Input Options	
Allow Wrong Button Failure	If true, when player Presses a button that's not the correct one, the QTE will fail. If false, the QTE persists until succeeded or times out.
Use Unity Inputs	Off ->, standard keyboard input is used On -> the script will use Unity's buttons/Axis defined in the Input manager instead.
Button # Key Press	If keyboard, type button name as defined by Unity here (under Button Names), if Inputs, type the name of the Input you want to use from the Input manager.
Random # Button	Check true if you want to use an Randomly Generated button, if true the "Key Press" fields disappear
Array of Random Buttons	Only appears if Random Button is checked true, If Array is left at 0, a random button will be chosen from all available buttons. If filled up with names of Keys or Inputs, a random one will be chosen from that list instead.
Timer Options	
Delay Length	The time in seconds that delays the entire QTE from happening once triggered.
Timer:	The time in seconds the player must press the button before failing.
Input Delay Length	The time in seconds a delay between when the QTE appears, and when Input starts being detected. This allows players a brief reaction time, otherwise players could trigger a QTE and then auto-fail it because they are in the middle of doing something.
Timer Per Button	Only during multi-button QTEs, check true to set a timer for individual buttons to disappear, note that the first "Timer" acts as a terminator (when it finishes all buttons will disappear)
Display Options:	
Hidden	If true, no On screen UI sprites are displayed, but the QTE will still happen, useful for situations like a Fight sequence where you can use the Animations of the enemy to give the player the Cue to press something, or you wish to show your own custom objects instead
Fade In	If True the Button fades in
Fade in Time	The amount of time the button fades in
Shake	If True the button shakes back and forth
Shake Offset	If shake is true, this is how much the button will shake
Position Manually	If false, buttons will appear where they are as set in the Canvas, if true. Buttons will be moved.
X Position	Distance in units horizontally away from the center of the canvas

Y Position	Distance in units Vertically away from the center of the canvas

The following options are excusive to the Mashable QTE.



QTE_Trigger (Script)

Main Settings

Overide Canvas: None (Game Object)

QTE Type: Mash

Mode: Using Collider

Repeatable ☑

Input Options

Mashing Tolerance 0.01

Allow Wrong Button Failure ☑

Use Unity Inputs □

Button 1

Randomize □

Button 1 Keyboard Ke m

Timer Options

Disable Timer Fail ☑

Delay Length: 0

Timer: 2

Input Delay Length: 0

Display Options

Hidden □

Fade In □

Pulsating Speed 48

Pulsate Frequency: 0.05

Shake 1st Button □

Overide Button Position ☑

Button 1 Position: X 0 Y 0 Z 0

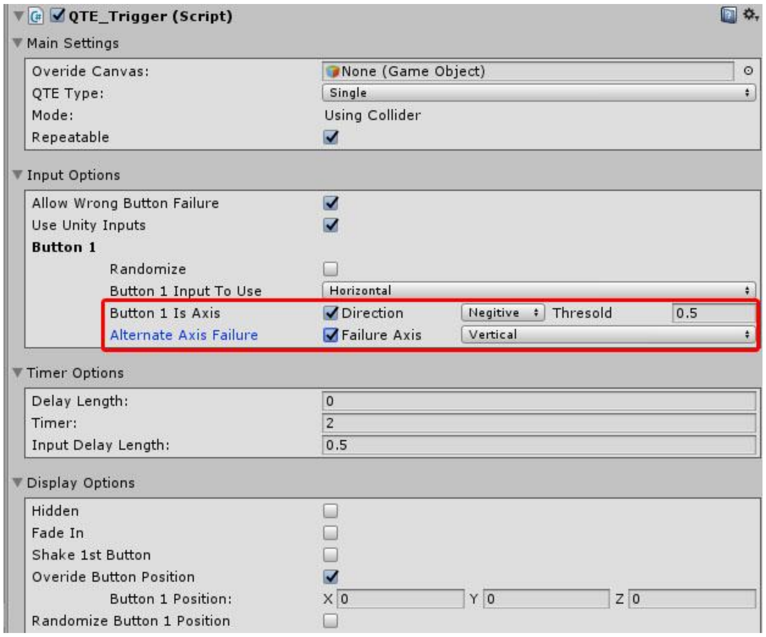
Randomize Button 1 Position □

Mashing Tolerance	There is a float value that starts with 0, every time the player presses the button it goes up by 0.2, but every frame the value of tolerance is subtracted from it, hence the "fighting" struggle. Player succeeds when value reaches 1. Increase tolerance to make the amount of button mashing harder to do, less to make it more easy.
Disable Timer Fail	If checked, the Mashing QTE will not automatically end after 'Timer' expires, instead the button remains until they succeed, and instead 'Timer' is used to fail the QTE if the player stops mashing the button.
Pulsating speed	How fast the button pulsates in and out, 0 will disable this effect.
Pulsating Frequency	The strength of the pulsating effect

Axis input

When using Unity's inputs from the Input Manager instead of straight up keyboard input, Axes are setup differently than simple buttons, As descried here Axes are a float value that ranges in between -1 to 1 rather than being a button that is "on" or "off".

Because of this difference, a workaround was implemented to detect the difference between the two directions.



QTE_Trigger (Script)

Main Settings

Overide Canvas: None (Game Object)
QTE Type: Single
Mode: Using Collider
Repeatable ☑

Input Options

Allow Wrong Button Failure ☑
Use Unity Inputs ☑
Button 1
Randomize ☐
Button 1 Input To Use: Horizontal
Button 1 Is Axis ☑ Direction: Negitive Thresold: 0.5
Alternate Axis Failure ☑ Failure Axis: Vertical

Timer Options

Delay Length: 0
Timer: 2
Input Delay Length: 0.5

Display Options

Hidden ☐
Fade In ☐
Shake 1st Button ☐
Overide Button Position ☑
Button 1 Position: X 0 Y 0 Z 0
Randomize Button 1 Position ☐

1. First you must check the "Button X is Axis" box if the input you wish to use is an axis
2. Choose which direction the Axis to detect "Positive" or "negative"
3. Set the Threshold the Axis must reach to trigger a success. This should not be a negative value or above 1.

Alternate Axis Failure

If the "Allow Wrong Button Failure" option is checked, then if the user inputs the opposite direction specified then the QTE will fail.

For example, if the Horizontal Axis in the Positive (Left) direction is chosen, going negative (Right, the opposite direction) will fail the QTE.

However, other directions, such as the vertical axis (up and down) will not fail the QTE, nothing will happen because that axis is not being detected. To enable this, check "Alternate Axis Failure" and choose the axis you also want to cause failure.

This way only going in the correct direction (left) will succeed, any other direction will fail the QTE.

Custom UI Images

You will find the images the QTE system uses as sprites inside the folder "\Resources\QTE_System\Textures".
You can replace the images in those folders with your own, there is no limitation on what size they are, and they can be in any image format readable by Unity (I have provided .psds).

It is required that the images and anything else in the "Resources" folder stay where they are because all of the images need to be included in the game for potential use by the system. If they are placed outside of the Resources folder and no gameobject in any scene is using them, then they are omitted when the game is built and therefore will cause errors when the game is running. But anything else added by this system is free the move about the project folders.

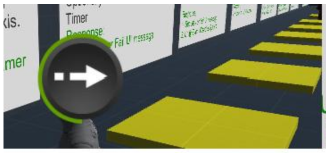
Naming Conventions:

The images have a naming convention: "QTE_(Button or Input name)" For example if you want to create a button for the keyboard key "[", create a new image named "QTE_["
If you type in a name of a button/key in a Triggering script that there is no image for, No image will appear on screen, and a Debug error message will tell you what Key/input is missing an image.

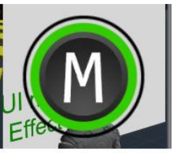
Displaying Custom Objects

You can prevent the UI images from appearing on screen by checking the "Hidden" property in Triggering scripts. This sets the onscreen UI images to invisible, but the Quick Time Event still fires anyway.
You can then parent your own objects to the Gameobject Buttons In the canvas and they will appear instead.

Visualizations



Wiping Circle



Expanding Circle

You can visualize the timer of a QTE as it is counting down by having a wiping circle that changes color behind it.

1. Drag in the "QTE_TimerUI" prefab into your canvas from "/QTE_System/Prefabs" (you only need to add it if the canvas dose not have it already)
2. Attached "QTE_Response_TimerUI.cs" to your triggering gameobject.

"QTE_Response_TimerUI.cs" has a "Position Offset" property that will offset the Timer UI's from the Buttons

If you are finding that the Timer UI circles are drawing overtop of the Buttons, You can attach a canvas component to the Timer UI Gameobjects and set "Overiding sorting" to true, then "Order in layer" to -1

You can also visualize the struggle/progress of the mashing QTE by having a Circle expand outward from the button until it reaches an outside ring

1. Drag in the "QTE_MashUI" prefab into your canvas from "/QTE_System/Prefabs" (you only need to add it if the canvas dose not have it already)
2. Add "QTE_Response_MashUI.cs" to your triggering Gameobject

You can find the explanation of "QTE_Response_MashUI" properties here.

World Space Billboards

If you prefer the buttons to actually be existing in the 3D world, rather than 2D UI sprites onscreen, you can easily set this up.

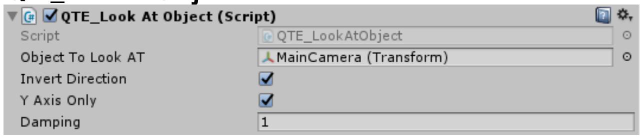
1. Create a World Space Canvas in the scene, and drag another “QTE_UI” prefab onto it.
2. Move the canvas/buttons to where you would like them to be in the scene
3. In a triggering script, Assign the new canvas as the Override Canvas

Because you’ve specified an override canvas, it will be used instead of the default canvas.

If you wish for the buttons to turn to look at the player/another object, I have included a script called “QTE_LookAtObject.cs”
Attach this script to each of the Button UI Images, and assign what object to look at

If you wish for the buttons to travel along with other objects in the scene, I have provided a script called “QTE_ButtonTravel.cs”.

QTE_LookAtObjects.cs

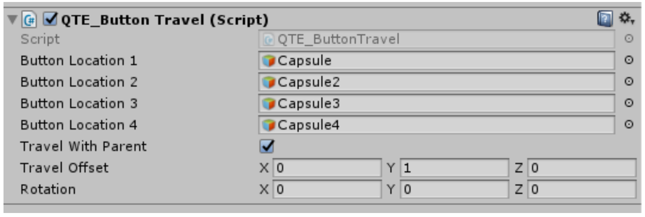


☑ QTE_Look At Object (Script)

Script	QTE_LookAtObject
Object To Look AT	MainCamera (Transform)
Invert Direction	☑
Y Axis Only	☑
Damping	1

Object to look at	Assign a gameobject that the buttons will face towards
Invert Direction	If the buttons show up facing backwards, flip this option to fix it.
Y Axis only	Buttons only rotate around the Y axis to face the object, otherwise the rotate fully
Damping	Use this value to create a lag effect so the buttons rotate slowly to look at the object.

QTE_Button_Travel.cs



☑ QTE_Button Travel (Script)

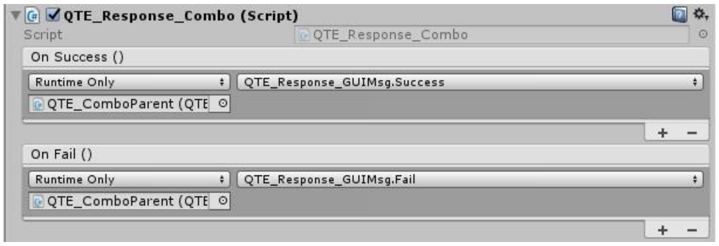
Script	QTE_ButtonTravel		
Button Location 1	Capsule		
Button Location 2	Capsule2		
Button Location 3	Capsule3		
Button Location 4	Capsule4		
Travel With Parent	☑		
Travel Offset	X 0	Y 1	Z 0
Rotation	X 0	Y 0	Z 0

Attach this script to the World Space Canvas you are using.

Button Location 1-4	Assign the gameobjects that you want the buttons to follow
Travel with Parent	Use this option to have the button "Travel" along with its supplied gameobject, If false the buttons will remain in place.
Travel Offset	The distance away from the location you want the traveling buttons to be.
Rotation	The rotation of the buttons

QTE Combos

You can create a combo of Quick Time Events that the player must complete all in a row to succeed rather than individually.



QTE_Response_Combo (Script)

Script: QTE_Response_Combo

On Success ()

Runtime Only | QTE_Response_GUIMsg.Success

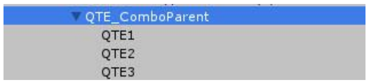
QTE_ComboParent (QTE

On Fail ()

Runtime Only | QTE_Response_GUIMsg.Fail

QTE_ComboParent (QTE

1. Attach to a gameobject "QTE_Response_Combo.cs", this script is now be the script you want to call to start the combo, rather than calling individual quick time events.
   Just like QTE_Trigger.cs, If the gameobject it's attached to has a collider set to "Is Trigger" the combo will fire if the player walks into it, otherwise you can call "StartCombo()" in the script to start the combo.
2. Create as many new QTE_Triggers as you want to be used in the combo, and make them children of the gameobject that has "QTE_Response_Combo.cs". They will be fired in order according to their sorting.

   QTE_ComboParent
   QTE1
   QTE2
   QTE3
3. From there, you can specify what happens in response to the success/failure of the combo using the two Unity Events

Response scripts

The triggering scripts will cause the QTE to fire, and the Main script will display the UI and perform the logic to see whether they failed or succeeded. But now you obviously want something to happen in your game "in response" to the result of the Quick Time Event, that's what the Response scripts are for.

- Response scripts attach to the same GameObject that has a QTE Triggering script.
- The response will only happen to that Quick Time Event that is Triggered by that object, and not by any other Quick Time Events in the scene.
- Multiple response scripts can be attached to have more than one thing happen.

What you want to have happen in your game after a Quick Time event is highly dependent on what exactly you want to accomplish in your game, I have included several example response scripts to get you started, but more than likely you may have to end up modifying the scripts, or writing your own responses to perform exactly what you want to do in your game.

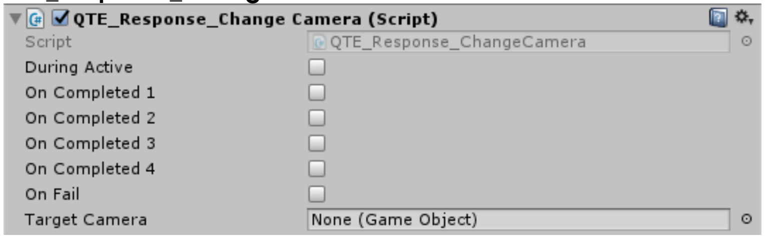
I have included a Template script "QTE_Response_Blank.cs", this script can be used to write your own response scripts.
You can perform an action

- While the QTE is happening (e.g. Disable player input)
- A different action per each success button
- A different action when the QTE fails by Timing out, or by wrong button press.

If you don't intend to use the Multi-button QTEs with your response script, then the "if" statements for the different successes can be left empty, or deleted entirely.

Example Response scripts:

QTE_Response_ChangeCamera.cs



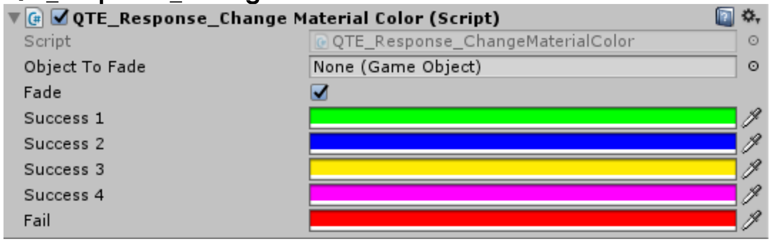
QTE_Response_Change Camera (Script)

Script: QTE_Response_ChangeCamera
During Active ☐
On Completed 1 ☐
On Completed 2 ☐
On Completed 3 ☐
On Completed 4 ☐
On Fail ☐
Target Camera: None (Game Object)

Turns off all cameras in the scene, and turns on the one specified. Cameras need to be tagged as "Main Camera"

Target Camera	The Camera Gameobject to switch to.
During Active	Changes to the camera only while the QTE is actively being shown.
On Completed #/Fail	Changes to the camera upon success/fail.

QTE_Response_ChangeMaterialColor.cs



QTE_Response_Change Material Color (Script)

Script: QTE_Response_ChangeMaterialColor
Object To Fade: None (Game Object)
Fade ☑
Success 1 🟩
Success 2 🟦
Success 3 🟨
Success 4 🟪
Fail 🟥

Changes the color of the material on a specified GameObject.

Object to Fade	The GameObject to change material on, if Nothing is specified, the Material on the Attached GameObject is used instead.
Fade	Fades out the material to invisible once it appears.
Success/Fail # Color	Colors to use.

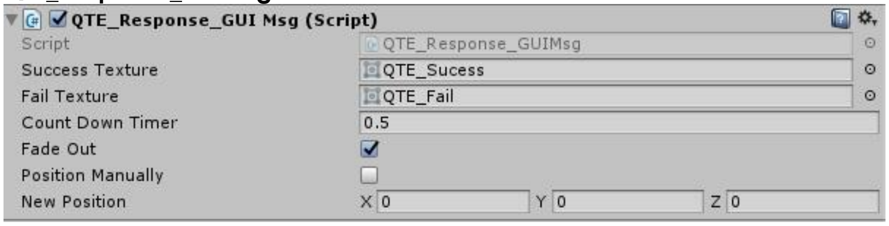
QTE_Response_DisablePlayer.cs

Disables the player from moving

This script is only be used only with the Third Person character that comes in the example scenes, which is from unity's example assets.
Provided as an example of disabling the player. If you are not using Unity's Third Person character you can remove this script from the project.

In order to disable the player movement, I've had to modify "ThirdPersonUserControl.cs" that comes with Unity's standard assets to have a Boolean DisableMovement, that allows me to do so.

QTE_Response_GUIMsg.cs



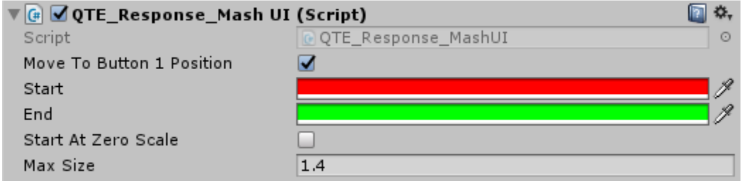
☑ QTE_Response_GUI Msg (Script)

Script	QTE_Response_GUIMsg
Success Texture	QTE_Sucess
Fail Texture	QTE_Fail
Count Down Timer	0.5
Fade Out	☑
Position Manually	□
New Position	X 0 Y 0 Z 0

Displays a Success or Fail image

Success Texture	The Sprite to display when the player succeeded the QTE
Fail Texture	The Sprite to display when the player failed the QTE
Count Down Timer	How long the Image remains on screen
Fade out	If true, the texture fades out over time.
Position Manually	Override the position of the image
New Position	The position to move the image to.

QTE_Response_MashUI.cs



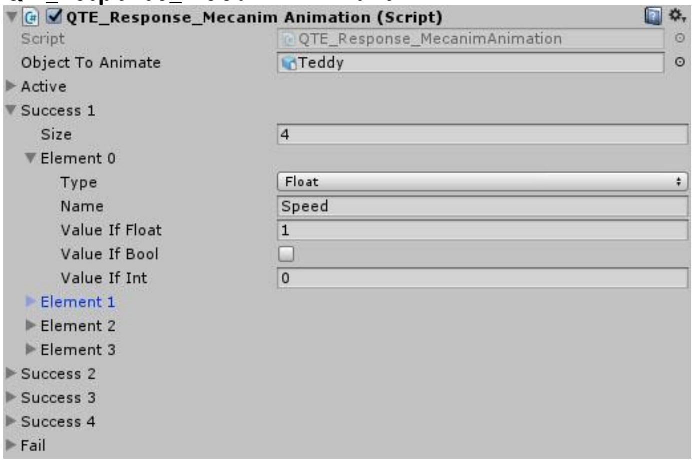
☑ QTE_Response_Mash UI (Script)

Script	QTE_Response_MashUI
Move To Button 1 Position	☑
Start	🟥
End	🟩
Start At Zero Scale	□
Max Size	1.4

Displays a visualization of the mashing. A circle will expand and change color out from the QTE Button until it reaches a ring outline.

Move to Button 1 Position	Move the mashing UI images to the same location as the QTE button
Start	The color of the expanding circle at the start
End	The color of the expanding circle when the mashing is completed
Start at zero scale	By default the expanding circle is the same size as the QTE button circle, and expands from there, check this option to cause the expanding circle to start from 0 salce
Max size	The maximum size the circle can expand to relative to the size of the QTE button.

QTE_Response_MecanimAnimation



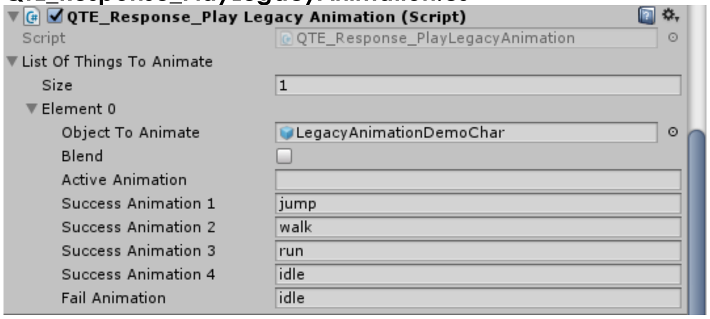
☑ QTE_Response_Mecanim Animation (Script)

Script	QTE_Response_MecanimAnimation
Object To Animate	Teddy
▶ Active	
▼ Success 1	
Size	4
▼ Element 0	
Type	Float
Name	Speed
Value If Float	1
Value If Bool	☐
Value If Int	0
▶ Element 1	
▶ Element 2	
▶ Element 3	
▶ Success 2	
▶ Success 3	
▶ Success 4	
▶ Fail	

Changes parameters of an Animator controller on a GameObject

Object to Animate	The GameObject to animate
Active, Success 1-4, Fail	An Array of values, Add as many as you want for the amount of Animation Controller Parameters that you want to change per response condition.
Type	Choose what Type of Parameter it is.
Name	The name of the Parameter
Value If __	Value to change according to the "Type" you picked above.

QTE_Response_PlayLegacyAnimation.cs



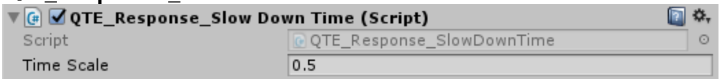
☑ QTE_Response_Play Legacy Animation (Script)

Script	QTE_Response_PlayLegacyAnimation
▼ List Of Things To Animate	
Size	1
▼ Element 0	
Object To Animate	LegacyAnimationDemoChar
Blend	☐
Active Animation	
Success Animation 1	jump
Success Animation 2	walk
Success Animation 3	run
Success Animation 4	idle
Fail Animation	idle

Plays an animation on a specified GameObject. (specifically to be used with legacy animations pre Unity 4.0)

Object to Animate	The GameObject with attached "Animation" component to animate;
Blend	Weather to use animation.Blend, instead of animation.Play
Active/Success/Fail Animation #	The name (string) of the animation to play back.

QTE_Response_SlowDownTime.cs



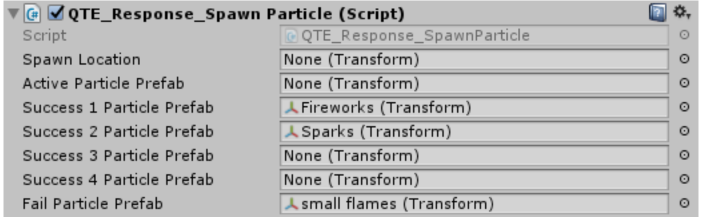
☑ QTE_Response_Slow Down Time (Script)

Script	QTE_Response_SlowDownTime
Time Scale	0.5

While the QTE happens, time slows down

Time Scale	The factor in which times slows down. 1 = Normal speed.

QTE_Response_SpawnParticle.cs



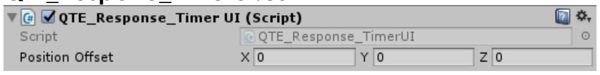
☑ QTE_Response_Spawn Particle (Script)

Script	QTE_Response_SpawnParticle
Spawn Location	None (Transform)
Active Particle Prefab	None (Transform)
Success 1 Particle Prefab	Fireworks (Transform)
Success 2 Particle Prefab	Sparks (Transform)
Success 3 Particle Prefab	None (Transform)
Success 4 Particle Prefab	None (Transform)
Fail Particle Prefab	small flames (Transform)

Spawns a particle

Spawn Location	The location to spawn the particles at, if None, spawns in the same location as the Attached GameObject.
Active Particle prefab	The Particle prefab to spawn While the QTE is happening.
Success # Particle prefab	The Particle prefab to spawn when the player succeeds the QTE, based upon what button got chosen, 2-4 can be left blank if not using the multi-button QTEs
Fail Particle prefab	The Particle prefab to spawn when the player Failed the QTE.

QTE_Respone_TimerUI.cs



☑ QTE_Response_Timer UI (Script)

Script: QTE_Response_TimerUI

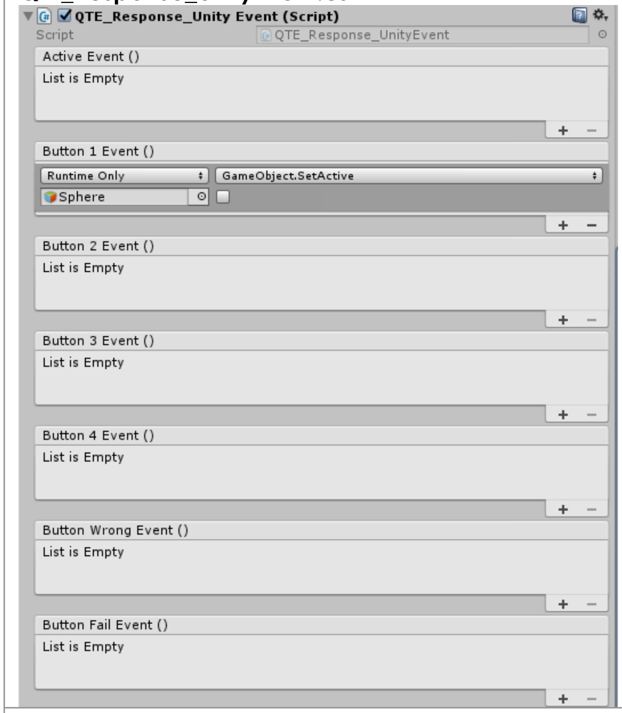
Position Offset: X 0 Y 0 Z 0

Creates a Visualization of the timer behind the QTE Button

A colored circle around the QTE button radial wipes away and changes color as the QTE timer counts down.

Position Offset	Reposition the circle relative to the QTE Button

QTE_Response_UnityEvent.cs



☑ QTE_Response_Unity Event (Script)

Script: QTE_Response_UnityEvent

Active Event ()
List is Empty

Button 1 Event ()
Runtime Only | GameObject.SetActive
Sphere ☐

Button 2 Event ()
List is Empty

Button 3 Event ()
List is Empty

Button 4 Event ()
List is Empty

Button Wrong Event ()
List is Empty

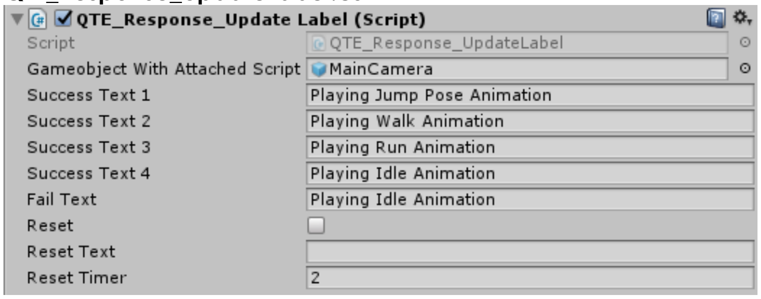
Button Fail Event ()
List is Empty

Fires Unity Events

A set of unity events for each QTE Condition.

You can fill up the events with gameobjects in the scene and select from the dropdown all the various actions you can call on that object.

QTE_Response_UpdateLabel.cs



☑ QTE_Response_Update Label (Script)

Script: QTE_Response_UpdateLabel

Gameobject With Attached Script: MainCamera

Success Text 1: Playing Jump Pose Animation

Success Text 2: Playing Walk Animation

Success Text 3: Playing Run Animation

Success Text 4: Playing Idle Animation

Fail Text: Playing Idle Animation

Reset: ☐

Reset Text:

Reset Timer: 2

Updates a script that is displaying a GUIlabel onscreen GameObject with attached script.	The GameObject with attached "GUI_Text.cs" script, that is putting the GUILabel onscreen
Success/Fail Text #	The text (string) to display
Reset	Resets the text after a timer completes
Reset Text	The text to reset to
Reset Timer	The time in seconds until reset.

Legal

Purchase of this Quick Time event system through the Unity asset store is covered by Unity's Terms of Service and EULA, you can view that information here.

http://unity3d.com/company/legal/as_terms

Contact:

mattswanton3d@gmail.com
http://www.mattswanton.com

Unity Forum Topic
http://forum.unity3d.com/threads/188088-Quick-Time-Event-system

Change List:

Version 1.0 - June 24, 2013

- Initial Release

Version # 1.1 - July 1st 2013

- Unity 3.5 version released
- lowering required version to base Unity 4 instead of 4.1
- Added Response script for a Slow down "bullet time" effect while the QTE is happening
- Fixed a bug with the Cutscene scripts auto-firing when the scene started

Version # 1.2 - July 8th, 2013

- "Enable Button Fail" option added, so that the QTE will not fail if wrong button is pressed.
- Triggering by Animator Curves support added.
- Adding ability for response scripts to detect & respond differently the two different kinds of failures (Wrong button, or timeout)

Version # 1.3 - August 28th, 2013

- Combined the 10 triggering scripts into one single script
- Triggering script has its own Custom Editor Inspector script
- references to "Cutscene" functionality, now referred to as "Manual"
- Individual timers for multi-button QTEs
- QTE Delay Timer (Different from the Input Delay Timer)
- Change camera response script
- Fixed 2 bugs with Billboards script, will now work in Manual (Cutscene) mode, and now no longer has to be above the trigger script.

Version # 1.4 - December 29th, 2013

- The main script now has a public function you can call "InteruptQTE()" that will halt and cancel out any QTEs that are playing.
- Most of the update() based timers have been replaced with Coroutine yield timers, I believe these are more efficient.
- Fixed a Null error bug when using the Delay timer with multi-button QTEs

Version 1.5 – May 2017

- Entire system updated to use Unity 5's new uGUI system, old legacy GUI support dropped. Minimum required version is now Unity 4.6
- Main QTE script is now a singleton, better way of accessing it rather than constantly using Gameobject.Find() in every script that needs to access the main script.
- General optimization overhaul of the code base. (for example base class, using arrays instead of separate declared variables, Cleaner syntax for response scripts (old scripts still work), other misc. cleanups)
- System now no longer Instantiates and Destroys objects, Necessary gameobjects are set up in the scene at the start, and when no longer needed they are disabled. Far more memory efficient.
- New demo scene
- New example UI sprites
- QTE Combo script added to add combo functionality

- In the triggering script, you no longer have to type in a name to use a button defined in the Input manager, instead a drop down list is presented
- Now any axes can be used in a QTE, not just the vertical/horzitonal axes, and you can choose the threshold the axis must reach to succeed.
- Along with choosing an any axis, you can choose another axis to also fail the QTE,
- Wiping Circle Timer Visualization added
- Fade in buttons option added
- Mashing Visualization
- Bug Fixes:
  -Calling QTE manually fixed
  -World space UI buttons now can shake and pulsate, previous method of using mesh billboards did not.
  -Fixed a bug where odd behavior would happen after a mashable QTE was completed (Repsonses triggering twice for example)
  -Fixed a bug where Multi-timer QTEs Buttons would stop disappearing correctly
  -Fixed a bug where Mult-button QTEs would sometimes disappear if a multi-timer QTE was completed previously.
  -When the randomize option is checked, it will now also choose axes along with buttons.
  -QTE triggers that are prefabs now retain their modified settings when hitting play.