

Torque Game Builder – Input Interaction

Introduction

Input is a very important factor to any game. We can't have a game unless we can let the player interact with the game. Fortunately with Torque Game Builder (TGB) we have two very easy and expansive ways to interpret input interaction. These two ways are the following:

1. Mouse Event Callbacks
2. ActionMaps

These two methods handle input interaction from typing keys on your keyboard, to moving your mouse in your TGB game and clicking, and even to moving a joystick. This reference will cover and explain both of these types of interactions as well as including full examples in utilizing them.

Quick Reference

Mouse Event Callbacks Reference

Mouse Event Callbacks			
Callback	Synopsis	Parameter(s)	Detail
onMouseDown()	Left Mouse Button Pressed.	(this, modifier, worldPosition\$, mouseClicks)	this – Current Object (self) modifier – Alt/Ctrl/Shift Key modifiers used. worldPosition\$ – Position in world-coordinates. mouseClicks – Number of mouse clicks used. NOTE:- The coordinates returned have been converted to world-coordinates and are automatically adjusted for the camera move/zoom. You can use these coordinates directly with all other TGB objects and also use it for object picking (seeing <i>pickPoint</i>).
onMouseUp()	Left Mouse Button Released.	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onMouseMove()	Mouse has been moved.	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onMouseDragged()	Mouse has been dragged (Left Button)	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onMouseEnter()	Mouse has entered the <i>t2dSceneWindow</i> .	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onMouseLeave()	Mouse has left the <i>t2dSceneWindow</i>	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onRightMouseDown()	Right Mouse Button Pressed.	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onRightMouseUp()	Right Mouse Button Released.	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).
onRightMouseDragged()	Mouse has been dragged (Right Button)	(this, modifier, worldPosition\$, mouseClicks)	(See “onMouseDown() <i>”</i>).

Mouse Modifier Callback Values			
Modifier Value	What the value represents	Modifier Value	What the value represents
1	Left Shift	2	Right Shift
4	Left Control	8	Right Control
16	Left Alt	32	Right Alt
Any other numbers are a combination of these, for example if you press Left Shift and Left Control together it will produce the modifier value of 5, or if you press Left Alt and Right Alt together it will produce the modifier value of 48.			

Torque Game Builder – Input Interaction

ActionMaps Reference

ActionMaps			
Script Example	Synopsis	Parameter(s)	Detail
<code>new ActionMap(testActionMap);</code>	Create a New ActionMap	(Name)	Use this in script to create a new ActionMap, replace "testActionMap" with whatever name you want to use.
<code>testActionMap.bind(keyboard, "u", toggleU);</code>	Bind a command using bind()	(device, event, function)	You can bind using this .bind() command, you must specify a device, the action to bind, then a function to be called when the event happens. The function specified must be set to receive a single value passed (ex. "function toggleU(%val)")
<code>testActionMap.bindCmd(keyboard, "u", "onUDown();", "onUUp();");</code>	Bind a command using bindCmd()	(device, event, script, script)	This function is similar to the .bind() command, except you pass two script values instead of just function names. The first script value will happen when the event is activated (on key down), the second when the event is broken (on key released).
<code>testActionMap.push();</code>	Activate an ActionMap	()	This will activate the ActionMap. Keep in mind the most recently pushed ActionMap will receive the events if multiple ActionMaps are pushed with the same events set.
<code>testActionMap.pop();</code>	Disable an ActionMap	()	This will disable the ActionMap.

ActionMap Available Events to Bind			
Keyboard General Events			
Event Name	Event Name	Event Name	Event Name
<i>Note: All general keys can be bound by simply using the key... ex. "u" will trigger the u key response.</i>			
backspace	end	win_apps	tilde
tab	home	cmd	minus
return	left	opt	equals
enter	up	lopt	lbracket
shift	right	ropt	rbracket
ctrl	down	numlock	backslash
alt	print	scrolllock	semicolon
pause	insert	rshift	apostrophe
capslock	delete	lcontrol	comma
escape	help	rcontrol	period
space	win_lwindow	lalt	slash
pagedown	win_rwindow	ralt	lessthan
pageup			
Keyboard Numpad Events			
Event Name	Event Name	Event Name	Event Name
numpad0	numpad5	numpad9	numpadminus
numpad1	numpad6	numpadadmult	numpaddecimal
numpad2	numpad7	numpadadd	numpaddivide
numpad3	numpad8	numpadsep	numpadenter

Torque Game Builder – Input Interaction

<i>ActionMap Available Events to Bind</i>			
numpad4			
<i>Keyboard Function Key Events</i>			
Event Name	Event Name	Event Name	Event Name
f1	f7	f13	f19
f2	f8	f14	f20
f3	f9	f15	f21
f4	f10	f16	f22
f5	f11	f17	f23
f6	f12	f18	f24
<i>Joystick/Mouse Events</i>			
Event Name	Event Name	Event Name	Event Name
button0	button8	button16	button24
button1	button9	button17	button25
button2	button10	button18	button26
button3	button11	button19	button27
button4	button12	button20	button28
button5	button13	button21	button29
button6	button14	button22	button30
button7	button15	button23	button31
<i>Joystick/Mouse Axes</i>			
Event Name	Event Name	Event Name	Event Name
xaxis	zaxis	ryaxis	slider
yaxis	rxaxis	rzaxis	
<i>Joystick POV</i>			
Event Name	Event Name	Event Name	Event Name
xpov	dpov	xpov2	dpov2
ypov	lpov	ypov2	lpov2
upov	rpov	upov2	rpov2
<i>Miscellaneous Events</i>			
Event Name	Event Name		
anykey	nomatch		

Torque Game Builder – Input Interaction

Mouse Event Callbacks

What is a Mouse Event Callback?

Quick Definition:

A mouse event callback is simply the response from the TGB engine when a mouse event has happened. This includes moving, dragging, clicking, releasing, leaving, and entering the screen so you can easily handle this input in your game a specific way.

In-Depth Definition:

In TGB, mouse events are Scene Window specific. That way you can handle multiple Scene Windows in different ways if you like. So when you either move, drag, click, release, enter the screen, or leave the screen with your mouse it will send what is referred to as a “callback” to the Scene Window. Callbacks are basically preset function names that the engine will call when events happen. This allows you to interact with the engine upon events. Not only does this interaction trigger the callback method, but it passes plenty of useful data to the function, such as the position where the event happened, the click counts if the mouse was pressed as well as any other information that could be useful.

How do we use Mouse Event Callbacks?

Using mouse event callbacks is very simple, since the callbacks are done through a Scene Window we must first find out the name of the Scene Window we are using. The Scene Window is the actual GUI object that the game is displayed on, the default Scene Window used is called “SceneWindow2D.” Once we have our Scene Window name we can add some simple script function calls like these:

```
function sceneWindow2D::onMouseMove( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onMouseDragged( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onMouseDown( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onMouseUp( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onMouseEnter( %this, %modifier, %worldPos, %mouseClicks )
{
}
```

Torque Game Builder – Input Interaction

```
function sceneWindow2D::onMouseLeave( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onRightMouseDragged( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onRightMouseDown( %this, %modifier, %worldPos, %mouseClicks )
{
}

function sceneWindow2D::onRightMouseUp( %this, %modifier, %worldPos, %mouseClicks )
{
}
```

You can see the pattern of function names that represent each mouse event. There are three different and very useful values passed to these callbacks, the first is called “%mod” which represents the Alt/Ctrl/Shift modifier keys if used, the quick reference modifier table explains what these values are output as. The “%worldPos” represents, what is probably obvious, the world position of the mouse event. The final value is fairly evident as well, “%mouseClicks” which represents the amount of mouse clicks used.

With these methods you can pretty much insert any functionality you need there, just remember that “sceneWindow2D” used in these functions is just the default Scene Window, if you create more Scene Windows then you will need to add the proper functions, like these, for that new Scene Window name.

ActionMaps

What is an ActionMap?

Quick Definition:

An ActionMap is how we can receive “actions” or input events from TGB. These actions can be anything from pressing the space bar, up arrow, or “w” key to moving the mouse horizontally or vertically.

In-Depth Definition:

An ActionMap is an object that you can define from script. These ActionMaps hold key/input binds that give you event's when certain input events happen. You can “push” to enable these ActionMaps and “pop” to disable them. This works almost as a layer system, that way when multiple ActionMaps specify commands to be triggered on an event, the most recently pushed one will be the event triggered. Multiple ActionMaps can be enabled at the same time, you can then “bind” input commands to an ActionMap in one of two different ways. One method specifies two functions, one that will be called when the event has occurred (when a key is pressed), the other when the event is broken (when the key is let up). The other method you only specify one function, this function will be called passing a value. If that input

Torque Game Builder – Input Interaction

command bound is a key that can be pressed and released then it passes a true or false (a 1 or 0) which signifies being pressed and released. Further uses for this method of binding a command is used when input commands don't send back a simple pressed and released event, but a value, for example the mouse. This way you can accomplish more complex workings with input commands.

How do we use ActionMaps?

Using ActionMaps is fairly simple, first we must create a new ActionMap in script, like this.

```
new ActionMap(testActionMap);
```

Now we have a new ActionMap object called “testActionMap.” We can now call one of two different functions on this object to “bind” keys/events to this ActionMap.

Using the .bind() command.

The first command is .bind(). When you use this command to bind an action you must specify the input device you are wanting to bind an event on (such as the “keyboard” or “mouse”), then the event it will bind, as well as the function it will call when the event is processed. Here is an example of binding it to the “u” key event.

```
testActionMap.bind(keyboard, "u", toggleU);
```

Now we have bound a “u” key to our testActionMap with the “toggleU” function to be called when the event is processed. Now we can set up the toggle function like this.

```
function toggleU(%val)
{
  if(%val)
  {
    echo("u key down");
  } else
  {
    echo("u key up");
  }
}
```

One very important thing to note is that this toggleU function is set to receive a “%val” value. This is very important, when using the .bind() command you must always set your functions up like this. In the case of a key (like our “u” key example) the %val will only ever be either a

Torque Game Builder – Input Interaction

1 (representing the key was pressed) or a 0 (representing the key was released). As you can see we can separate the functionality with a simple if statement.

An non-key example of using `.bind()` would be this.

```
testActionMap.bind(mouse0, "xaxis", xaxis);
```

As you may notice we aren't binding an event from the “keyboard” this time we chose “mouse0” which represents the first mouse (since TGB can handle input from multiple mice). The event we're binding is “xaxis” and we simply call an “xaxis” function. The “xaxis” basically represents the mouse movement in the xaxis. We can set up the xaxis function like this.

```
function xaxis(%val)
{
    echo("xaxis = " @ %val);
}
```

In this case we are simply displaying what the value passed to this xaxis function is. In the case of our mouse xaxis event the value passed can be a wide range of numbers from negative to positive, in this specific case the value represents the distance moved on this event since the last event, so if you move your mouse suddenly you will get large numbers sent and then it will become small, while if you move it softly then you will only get small numbers. This give you an extremely accurate way to measure mouse actions (if you aren't using your mouse callbacks on your SceneWindow). In most cases you won't need to bind the mouse actions to an ActionMap, mainly just keys.

Using the `.bindCmd()` command.

An alternative way to bind commands is using a different function called “`.bindCmd()`.” This function works fairly similar to the `bind()` function, except instead of setting a single function to be called when the event is processed, you set two functions. One will be called when the event is activated (on key down), while the other is activated when the event is broken (on key release). In this case you don't pass just a “function” though, you pass a string that represents script to be run when you activate that event. That means that you can pass a single line if you want to do a single script action, though generally you will simple pass a function call in a string. You can set up the same “u” key like this.

```
testActionMap.bindCmd(keyboard, "u", "onUDown();", "onUUp();");
```

Torque Game Builder – Input Interaction

Notice we wrapped the functions in a string as well as making it a full script call to that function. Now we can have one function do what needs to be done when the “u” key is pressed, while the other can do what is needed when the “u” key is released.

There is one final step that must be done to get the ActionMap to work. We've created it and called it testActionMap. We then bound multiple action events to it on multiple devices. The last step is to “push” the ActionMap so it will know to receive input. This is a very important step, it is also important to keep in mind that the most recently pushed ActionMap will take precedence over the previously set keys, this allows you to create very usable control schemes with multiple ActionMaps while enabling and disabling them easily. To push our ActionMap we can do this.

```
testActionMap.push();
```

This will activate our ActionMap and it will start receiving our action events, to disable an ActionMap we can use a simple function called “pop” like this.

```
testActionMap.pop();
```

This will disable our ActionMap, giving our previous ActionMap precedence again.