T-Scripts: Examples and Uses

By Rebel

Q. HOW DO I GET A STEGASAUR TO EAT?

A. All herbivores (including the stegasaur) must be changed to carnivore status. In order to do this, simply change the TScript value; int ArcheType = 1 to int ArcheType = 0 within the steg's TScript. The value, int ArcheType = -1 can also allow a steg to eat. Though a herbivore with int ArcheType = -1 tends to ignor the player, it will at times attack. (Please be certain that action bool ActEat = true is also listed within TScript Values)

Food Sources: Generally, any object labeled AIType = 3, which equates to bone, carcass, flesh, et cetera can be used. Mostly, in order to properly present the stegasaur as feeding, the food cube would be labeled invisible (bool Visible = false) and placed within certain target plants.

Q. HOW DO I GET A RAPTOR TO DRINK?

A. A water object needs to be tangible, (should be invisible and placed just beneath a body of water) and the value int AIType = 15 must be included within your water object's TScript values. A thin box works nicely as it can become a part of the water's basin and not interfer with either a dino or a player's walking action while in the water itself. Beyond being certain that the value bool Act Drink = true is included within a raptor's tscript values, there's nothing else to it except to add the 'raptor drink' audio trigger to your level which can be found alongside other vocal triggers in the tc level (basement area).

Q. HOW DOES A COLLISION TRIGGER WORK?

A. To understand what a collision trigger does and how it reacts, you simply need to understand the term, 'collision'. It can be a collision between two objects, just one object, or it can be non-tangible, meaning, the collision of sound. To better understand this, below is an example similar to the 'shooter' triggers found in the tc.level.

```
string Class = "CCollisionTrigger"
string Element1 = "RAPTOR-HEAD"
bool SoundMaterial1 = true
string Element2 = "BULLET"
bool SoundMaterial2 = true
int FireCount = 1
```

```
group Action00 = {
int AlphaChannel = 0
int ActionType = 0
string Sample = "VA141"
float Attenuation = 0.000000
float Volume = -5.000000
}
}
```

The above example uses sound to satisfy the trigger's conditions, which in this case is a 'bullet' sound impacting against the second sound, 'Raptor-Head'. If you are familiar with the dinosaurs in Trespasser then you may recall that raptor's have three particular string objects attached to them; a tail, body and head and they make up the tangible parts of the dino. Each of these string models have a sound assigned. In our example, the 'RAPTOR-HEAD' sound is attached to the value; string Element 1. Naturally, the other, our 'bullet' sound, is attached to string Element 2. Now, in order to satify the above trigger, these two sounds need to 'collide' with one another. Once this occurs, the groupAction00, which plays the string Sample "VA141" (Anne VoiceOver) is activated. (Note the actiontype '0' which corresponds to audio).

Some items to note: Whenever desiring sound to detect collision, the values bool SoundMaterial1 (and SoundMaterial2) must be entered into the tscript of the trigger. Having two SoundMaterials to satisfy the trigger isn't necessary, of course, as you can simply use one sound to do this. eg. Shot a gun, a bullet sound has been recorded and the trigger will fire.

Object Collision Example:

```
string Class = "CCollisionTrigger"
string Element1 = "Hammer"
string Element2 = "Door" (Door is in Frozen State)
int FireCount = 1

group Action00 = {
int ActionType = 10
string Target = "Door"
bool Frozen = false
}
```

Note, the omission of the SoundMaterials (1 and 2) since we desire this to be a 'physical' collision of two objects. In this example, our 'Door' is locked (or frozen) and cannot be opened until a collision between our example 'Door' and a 'Hammer' has occurred. In the group Action (00), this trigger points to an object (the door) and 'unfreezes' once trigger fires. Note the actiontype for this type of event, ActionType = 10 and the fact that not just any object would unfreeze this door, but only the 'Hammer'. Of course, any object can do the trick, this is just an example. (The script above also happens to create a puzzle too!)

Lastly, if you've noticed, collision triggers can be used to trigger a variety of events, specified of course, by your desires and the actiontype chosen.

Q. <u>I am using two location triggers. They are meant to be entered in order, but if a player reached the second trigger first how can I stop the other trigger from ever firing?</u>

A. We'll use an example for this:

```
group PlayerGotoLab = {
string OverlayText = "Hmm. Maybe I should try the lab.... "
int ActionType = 34
int FireCount = 1
string FireExpression = "!PlayerInsideLab"
PlayerEnterTrigger = true
string Class = "CLocatonTrigger"
}

group PlayerInsideLab = {
sound Sample = "VH12"
int ActionType = 1
int FireCount = 1
PlayerEnterTrigger = true
string Class = "CLocationTrigger"
}
```

Looking over the scripts above, they correspond to separate triggers. The first, giving a hint to the player by way of a textover that they should look elsewhere, the second, a voiceover sample, (so let us assume that the voiceover tells the player something of importance and that, was what they were looking for). Therefore, it is no longer desirable that the first trigger 'PlayerGotoLab', displays its hint if the player has already found the lab and has set off the trigger, 'PlayerInsideLab'.

To avoid becoming complicated here, just for argument sakes, let us say that the value inside of trigger 'PlayerGotoLab', string FireExpression = "!PlayerInsideLab" is monitoring whether or not the player has entered the Lab. So long as the player hasn't the hint will display since the FireExpression !(Not)PlayerInsideLab is actually a true statement and the trigger is still valid, or in an 'active' state. On the otherhand, if the player has reached the lab before the hint trigger, the expression is no longer a true statement and the trigger will not fire.

The above is perhaps rather confusing, but there are active examples that can be studied by using TresEd. One example, can be found inside the Ascent2 level (elevator) and another example of this scripting can be found in the Summit Level. Goto Trig_CircleFight at the top of the Summit (the raptor and CBoss location trigger) to look over some active scripts and perhaps gain a better understanding of their use. The Summit example you may note is directly tied into the easteregg 'Ozymandias'.

Q. <u>Herbivores move rather slowly, anyway to speed them up?</u>

A. The short answer is, yes.

TScript Value: float Speed = ?.000000 Consider default as 1, and the value is used in original dreamworks levels to slow the larger herbivores rather than speeding any dino up. The values can be added to any dinosaur tscript, though results vary. For example, using the value to increase a raptor's speed often results in the raptor literally tripping over itself. However, a moderate increase to the value of four-legged herbivore creates a decent result. (An example of this value used in tscript can be found in the lab level, the stegasaur)

Q. <u>I'm designing (editing) a level. How do I spawn a Raptor?</u>

A. Three things must exist: The Raptor, a trigger which would trigger the event and the destination, which is often a simple cube with type 1 Geometry.

Here's the Scripts:

```
group Trig_SpawnRaptor = {
string Class = "CLocationTrigger"
bool PlayerEnterTrigger = true
int FireCount = 1
int AlphaChannel = 0
int ActionType = 18
string ObjectName = "RaptorC-01"
string TeleportDestObjectName = "Teleport_RaptorToHere"
bool HeightRelative = true
bool SetPosition = true
bool SetOrientation = true
}

group Teleport_RaptorToHere = {
bool Visible = false
int ext_GeometryType = 1
}
```

As you have probably guessed by looking over the scripts, not much to it. Here, we are using a location trigger to initiate the teleport of RaptorC-01, though you can just as easily have used a Collision trigger to do this. To mimic the tnext cheat, you can teleport our Anne as well by simply inputting 'Player' as the variable in the string ObjectName.

Q. What is an Object Trigger?

A. As the name implies, it is a trigger which reacts to an as-signed object.

The three common values are: PickupObject, PutDownObject and UseObject.

Example:

```
group Trig_AnneGrabGun = {
string Class = "CObjectTrigger"
string A00 = "BigGun"
bool PickUpObject = true
int FireCount = 1
int AlphaChannel = 0
int ActionType = 0
string Sample = "VA21"
}
```

Quite simple, and self-explanatory. Any action can be assigned within the scripts, as well as the fact that as the value string A00 might have hinted to you, you can assign more than one object to initialize the trigger.

```
string A00 = "BigGun"
string A01 = "AnotherGun"
string A02 = "SomeOtherModel"
```

... and et cetera.

Q. I just imported two Raptors from pre-existing levels and they keep fighting each other. How can I stop that?!

A. Dinosaurs are assigned team values, ie. int Team = 15. To be certain that your desired dinos remain friends, simply edit their scripting accordingly. Their herbivore or carnivore status isn't important; providing they are assigned the same team, a Steg and a Rex would get along... Isn't that nice? 🙂

Q. <u>I just imported several Raptors and a TRex into my level but the Rex doesn't fend off the attacking Raptors. What's wrong?</u>

A. Depending upon which levels the dinosaurs were exported from, The Raptors may be missing their AITypes which identifies an object, such as attacking Raptors, as Dinosaurs. Easily rectified by being certain that the value; int AIType = 2 is present within the Dinosaur's $Models; Head, Body and Tail.

Q. **<u>While building my level, I have added some vegetation but I'm not hearing any sound effects while I walk thru them. What happened to the 'swoosh'?</u>**

A. The plants themselves do not generate any sound, that happens by way of invisible terrain_objects assigned a particular sound; ie. string SoundMaterial = "VEGE2"

Complete tscript as follows:

```
group TrnObj_Veg02-00 = {
string Class = "CTerrainObj"
int Height = 100
string SoundMaterial = "VEGE2"
float Culling = 10.000000
int ext_GeometryType = 2
}
```

The terrain_object itself is nothing more than a flat plane and a blank texture with a blank (black) opacity map which creates a transparent texture. You must place an instance (or copy) of the model, centered beneath each plant which you desire to produce a sound.

Q. <u>How can I create a distant gunshot sound by using triggers?</u>

A. Consider the TScript below a guide as to how you accomplish a sound (effects only) emitting from a distance.

```
group audiotrigger = {
group Action00 = {
string Sample = "GUN - RUGER REDHAWK 01"
int ActionType = 23
string Emitter = "SomeObject"
bool Attach = true
}
int FireCount = 1
bool PlayerEnterTrigger = true
string Class = "CLocationTrigger"
int ext_GeometryType = 2
}
}
```

'SomeObject', could be a rock, piece of wood, et cetera. Whatever object is chosen, it needs to be tangible. The further the object is in relation to its location trigger the more distant the sound will become. Using multiple actions, ie. action00, action01, and so on, you can have multiple sounds emitting from varied distance and/or directions.

(Thanks goes to Tomi for posing the question)

Q. How do you make a dead dinosaur?

A. You would use kill trigger settings, though use those values in a start trigger as shown below.

```
group Trig_KillDinoStart = {
string Class = "CStartTrigger"
int FireCount = 1
int ActionType = 17
string ObjectName = "SomeDino"
float HitPoints = 100.000000
}
```

Q. Can you make dinosaurs stay near other dinosaurs?

A. Of course! Any tangible object will do, and that includes dinos or even player. The partial script below is from the tc.level which holds the Baby Steg near its Mama.

```
group Baby = {
int Archetype = 1
int Team = 4
bool ActLookAround = true
bool ActStayNear = true
string StayNearTarget = "Steg-01"
float StayNearMax = 15.000000
float StayNearOK = 5.000000
}
```

(Question submitted by Second Illiteration)

Q. How does an animated control panel work?

A. The example below is a bit involved, though not particularly complicated. Taken from the TC.Level, the following is a breakdown of the models needed and the TScripts used.

This example is of the broken centrifuge controls. The models used, along with their associated triggers are;

* brokenbtn_on
* brokenbtn_off
* Broken_panel
* C_BrokenDummy

* TrigBroke_LightOff
* TrigBroke_LightOn
* TrigBroke_Sound
* TrigPlayerPoint-00
* TrigPlayerUnPoint-00

The first four listings are actual models which consists of an 'On' button, an 'Off' button, our broken display panel and a simple model strip consisting of faces which is textured with the additional textures required by the first three models mentioned. The models have only one texture assigned to them directly, but since the scriptings call for others this is where the texture strip (C_BrokenDummy) will serve its purpose as it is assigned all extra textures used. (In the level there is also a set of dummy buttons (on,off) as well, but for simplicity, let us assume that all additional textures are stored on this one texture strip)

So, let's break this down.. ..

```
group brokenbtn_on = {
int ext_GeometryType = 2
bool Visible = true
float Culling = 15.000000
bool CacheIntersecting = true
string Anim00 = "BTNONWH.BMP"
string Anim01 = "BTNONGR.BMP"
float Interval = -1.000000
int FreezeFrame = 0
}
```

Our first script here is our 'On' button. Though there are two animation strings attached to this model, present settings just does not allow the animation to run. The int FreezeFrame = '0' sets the model at its own texture (the texture model was imported with) and defaults the float Interval to '-1' which does not allow the animation to play without trigger intervention.

```
group brokenbtn_off = {
int ext_GeometryType = 2
bool Visible = true
float Culling = 15.000000
bool CacheIntersecting = true
string Anim00 = "BTNOFFWH.BMP"
string Anim01 = "BTNOFFGR.BMP"
float Interval = -1.000000
int FreezeFrame = 0
}
```

Our second script here is our 'Off' button. Again, this script contains the same settings as our 'On' button. Animation cannot play on without trigger intervention.

```
group Broken_panel = {
int ext_GeometryType = 2
bool Visible = true
float Culling = 15.000000
bool CacheIntersecting = true
string Anim00 = "PANELB.BMP"
string Anim01 = "panelml1.BMP"
string Anim02 = "panelml2.BMP"
string Anim03 = "panelml3.BMP"
string Anim04 = "panelml4.BMP"
string Anim05 = "panelml1.BMP"
string Anim06 = "panelml2.BMP"
string Anim07 = "panelml3.BMP"
string Anim08 = "panelml4.BMP"
float Interval = -1.000000
int FreezeFrame = 0
}
```

Our third script here is the actual control panel script and though the string animation textures (bmps) obviously differ, its animation cannot play without trigger intervention.

```
group C_BrokenDummy = {
int ext_GeometryType = 2
string Class = "CInstance"
bool Visible = false
bool Tangible = true
}
```

Our fourth script is attached to the texture strip. This contains all of the string Anim. 'bmp' files which the previous three scripts use during the play of their animations.

Setting the stage: Both buttons (On and Off) are dim, our control panel is dark. Naturally, we want to be able to change all that in order for these controls to react to us (as player).

Our first step would be to either create or copy the pointer triggers (Hand Point Triggers) whenever Anne has buttons to push,

but I won't cover this area as I'm certain most are aware of
these triggers already. Simply place a copy of these pointer
triggers so that they surround your targets, in this case, our
buttons.

Now, down to the fun stuff, the triggers. For simplicity purposes
I have combined the trigger, TrigBroke_LightOn with another
trigger, TrigBroke_Sound. We'll cover the Off position first --

```
group TrigBroke_LightOff = {
int ext_GeometryType = 2
string Class = "CLocationTrigger"
bool ObjectEnterTrigger = true
bool PointTrigger = false
int BoundVol = 1
string TriggerActivate = "$AnneHand+Anne"
float RepeatPeriod = 10.000000
int AlphaChannel = 0
group Action00 = {
int ActionType = 21
string Target = "brokenbtn_off"
int Frame = 1
float Interval = 0.700000
}
group Action01 = {
int ActionType = 23
string Sample = "SPEC-BUTTONS05"
}
group Action02 = {
int ActionType = 21
string Target = "brokenbtn_on"
int Frame = 0
float Interval = -1.000000
}
group Action03 = {
int ActionType = 21
string Target = "Broken_panel"
int Frame = 0
float Interval = -1.000000
}
}
```

The above trigger is actually a small cube, place directly on the
very top of our 'Off' button so that Anne's finger can actually in-
teract with it. Note, that this is a location trigger and can only
be activated by Anne's hand. Also note that there are four actions
within this TScript; Action00, Action01, Action02 and Action03. To
help explain matters, we'll look at all four.

Action00:

Our target within this action is the 'Off' button. The frame has switched from being frozen at '0' and is now instructed onto frame 1 at 0.770000 intervals. That's less than a second, and considering that there are only two frames associated with this button you will only notice the button light up for a brief second before reverting back to its original script settings. (On frame '0')

Action01:

Ah, an easy one. Simply play this effects sound sample.

Action02:

Our target within this action is the 'On' button. These settings simply mimic the object's original settings and are simple. Let's make sure that the 'On' button isn't lit.

Action03:

Our target within this action is the 'Control Panel'. Again, these settings mimic the object's original settings. Let's make sure our Control Panel is dark and not animated.

Alright, we've just covered what happens when player pushes the Off button, but what about the 'On'?

Well, below is the trigger script to handle that --

```
group TrigBroke_Sound = {
int ext_GeometryType = 2
string Class = "CLocationTrigger"
bool ObjectEnterTrigger = true
bool PointTrigger = false
int BoundVol = 1
string TriggerActivate = "$AnneHand+Anne"
float RepeatPeriod = 10.000000
int AlphaChannel = 0
group Action00 = {
int ActionType = 23
string Sample = "SPEC-BUTTONS05"
}
group Action01 = {
int ActionType = 23
string Sample = "SPEC-MAINFRAME BOOT START"
}
group Action02 = {
int ActionType = 21
string Target = "brokenbtn_off"
int Frame = 0
float Interval = -1.000000
}
group Action03 = {
int ActionType = 21
string Target = "brokenbtn_on"
int Frame = 1
float Interval = 5.200000
}
group Action04 = {
int ActionType = 21
string Target = "Broken_panel"
int Frame = 1
float Interval = 0.550000
}
}
```

The above trigger script (as previously stated) is for the 'On' button and animates the malfunctioning control panel. Again, our trigger is nothing more than a small cube placed directly on top of our 'On' button.

There are 5 total Actions involved here. The first two are easy ones, simple sound samples to play, so we'll just cover Actions, 02,03 and 04.

Action02:

Our target within this action is the 'Off' button. These settings simply mimic the object's original settings as explained earlier. Unlit, no animation --

Action03:

Our target within this action is the 'On' button. These settings instruct the animation to play its two frames, though rather slow-ly. The intervals between frames 1 and 2 is 5.2 seconds. Why this rather odd timed frame you may ask? We'll get to that in a sec. ;

Action04:

Our target within this action is the 'Control Panel'. These set-tings instruction the animation to play it's nine framed animation at an interval of 0.550000 seconds. As explained earlier, once an animation reaches its end it reverts back to its original settings again. (There are looped animations, but that's another topic)

Now, to explain the rationale behind the float intervals (or time between frames) of our 'On' button and the 'Control Panel'. To be simple about it, they are simply syncronized to one another. Even clearer, the 'On' button will stay 'On' for the entire duration of the Control Panel's malfunction animation. The varied durations in time allowing for both animations to end together.

There is, of course, more complex animation sequences than the example just given and a vast amount of varied arrangements than you can dream up, but regardless of complexity, you will find that the same values and techniques used here can be applied elsewhere.

(Thanks goes to Tomi (again!) for asking this question) 🙂